UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN TELECOMUNICACIÓN: SONIDO E IMAGEN
PROYECTO FIN DE CARRERA

SISTEMA DE REALIDAD AUMENTADA PARA APLICACIONES ANDROID

Autor: Natalia Mercedes Fernández Sánchez
Director: Jorge Muñoz
Tutor: Raúl Arrabales
Enero de 2012

Índice General

Índice de imágenes

Índice de código

Capítulo 1.

Introducción

1.1. Motivación del proyecto

La tecnología de realidad aumentada [1], consistente en la combinación de elementos virtuales con la imagen del mundo real, está cada día más presente en la vida cotidiana. Múltiples son las aplicaciones que se le están dando a esta tecnología, como en educación, en turismo o incluso en medicina.

Por su parte, desde la aparición de los *smartphones* o teléfonos *inteligentes*, la realidad aumentada ha visto incrementar sus aplicaciones notablemente. Obtener la información del entorno con *Layar* [2] o la de un usuario con *TAT augmented ID* [3], hasta conocer la valoración de los clientes sobre los restaurante de tu ciudad en *Yelp Monocle* [4], son solo algunas de las múltiples aplicaciones que día a día se están introduciendo en el mercado de los dispositivos móviles.

Un estudio de la consultora Gartner [5] prevé que la industria de los videojuegos en dispositivos móviles aumente entre un 15% y un 20% del 2010 al 2015, y es aquí donde la realidad aumentada ha empezado hacerse un hueco. Desde que los videojuegos en dispositivos móviles empezaron a tener mayor importancia, hasta hace relativamente poco tiempo, iPhone [6] era el único dispositivo capaz de soportar una buena gana de juegos con potentes funciones. Sin embargo Google hizo hincapié en que los dispositivos Android [7] pudieran soportar también grandes videojuegos. En un principio los desarrolladores tuvieron sus dudas, y en la aplicación de Android Market [8], los videojuegos eran bastante escasos, y aunque todavía los de iPhone generan la mayor recaudación, Android posee el 41% de la cuota del mercado, frente al 31% de Apple [9].

Estas circunstancias han motivado a la realización del proyecto. Crear un sistema de realidad aumentada, para una industria que crece día a día, en una plataforma que lucha por ser pionera en dispositivos móviles, y cuya principal aplicación está pensada para la creación de videojuegos. Por lo tanto el cometido de este proyecto es la realización de un sistema de realidad aumentada para el sistema operativo Android, que pueda ser utilizado para la creación de videojuegos, o aplicaciones similares. Dicho sistema será utilizado para la creación de un videojuego descrito en un anexo a la memoria, por lo que los objetos creados serán los correspondientes a dicho videojuego. El sistema es lo suficientemente independiente como para que cualquier usuario pueda diseñar sus propios objetos, e integrarlos en el sistema y que éste pueda ser utilizado en todas sus funcionalidades.

El sistema permite mediante la introducción de las coordenadas GPS [10] del usuario y de los objetos que se desean encontrar, que éstos sean visualizados por la pantalla del dispositivo móvil, cuando el usuario se encuentre a una determinada distancia de los objetos. De la misma manera, el usuario podrá capturar dichos objetos cuando se encuentre a una mínima distancia. Debido a que el sistema está pensado principalmente para su integración en videojuegos, permite al programador que la

utilice, poder dotar al juego de objetos ‘ítems’, y añadir texturas diferentes para que éstos tengan distintas funcionalidades dentro del videojuego.

1.2. Objetivos

El objetivo principal de este proyecto es la creación de un sistema de realidad aumentada para el sistema operativo Android con idea de que sea utilizado para la creación de un videojuego, pero que a su vez sea lo suficientemente versátil como para poder utilizarse en otro tipo de aplicaciones. De este objetivo principal podemos obtener los objetivos secundarios:

- Conocer las características que ofrecen los teléfonos móviles y como éstas pueden ser aprovechadas para la creación de un sistema de realidad aumentada:
    - Sensores: Android permite acceder a los sensores internos del dispositivo a través de varias clases contenidas en el paquete *Android.hardware.* Algunos de ellos, como el acelerómetro, se incorporaron desde la primera versión.

- Conocer la plataforma Android, que desde su anuncio en noviembre del 2007, se ha convertido en uno de los sistemas operativos para teléfonos inteligentes más importantes a nivel mundial. Su desarrollo ha sido tan rápido que desde la primera versión del API que apareció en 2008, se ha actualizado en 12 ocasiones.

- Estudiar la realidad aumentada, qué es, y qué aplicaciones tiene.

- Conocer el estándar OpenGL[11] , las opciones gráficas que ofrece, y como integrarlas en un proyecto, pues durante los 3 años de carrera no se ha trabajado nunca con este estándar.

- El sistema de realidad aumentada se integrará junto con otro proyecto para la creación de un videojuego, por lo que otro objetivo será realizar una buena planificación para coordinar la realización del proyecto con el videojuego donde se piensa utilizar el sistema de realidad aumentada, así como enfocar diferentes puntos del mismo para que puedan ser utilizados el videojuego concreto.

1.3. Estructura del documento

La presente memoria se divide en los siguientes capítulos:

Capítulo 1. Introducción y objetivos.

Se presenta una breve descripción del proyecto, haciendo un resumen de la evolución de los videojuegos para móviles en los últimos años, y los objetivos que han motivado realizar este proyecto.

Capítulo 2. Estado del arte.

Para poder ubicar este videojuego en la categoría de Realidad Aumentada, se presenta al lector en qué consiste dicha tecnología, de dónde viene, qué aplicaciones tiene y cómo ha ido evolucionando y por último, cómo puede utilizarse para la creación de videojuegos. Se hace también un breve repaso a la historia de los videojuegos y más explícitamente, a los videojuegos en los dispositivos móviles.

Capítulo 3. Desarrollo.

El cuarto capítulo se centra en el análisis, el diseño y la implementación del proyecto, exponiendo entre otros, los requisitos de usuario y requisitos software, los módulos de los que se compone el videojuego, los diagramas de clases y los detalles de la implementación de la parte del videojuego que nos ocupa.

Capítulo 4. Conclusiones.

Se exponen las conclusiones obtenidas tras la realización final del proyecto.

Capítulo 5. Trabajos futuros.

Se hace una reflexión sobre la posibilidad de incluir mayor funcionalidad al videojuego y de la oportunidad de cómo este puede extenderse.

Al final de esta memoria se encuentran las referencias y la bibliografía que han sido necesarias para su realización, así como tres anexos: planificación, presupuesto y descripción del videojuego *Invasión Androide*.

Capítulo 2.

Estado del Arte

Para ubicar tecnológicamente este proyecto, hablaremos sobre qué es la realidad aumentada. Todo el mundo hemos oído alguna hablar de éste término, ¿pero sabemos lo que significa?

En el apartado 2.1 se hace una introducción sobre qué es realmente este sistema. En el 2.2 haremos un repaso de dónde viene y en el 2.3 cuáles son algunas de sus principales aplicaciones.

En el apartado 2.4 nos centraremos en el principal cometido de este proyecto, analizando cómo han ido evolucionando los videojuegos hasta convertirse en lo que son hoy en día, cómo se han ido introduciendo en el mercado los juegos que hacen uso de la realidad aumentada y cómo ha ido evolucionando la tecnología hasta llegar a ellos.

2.1. Qué es la Realidad Aumentada

La Realidad Aumentada (del inglés Augmented Reality), es el término que usamos para definir las tecnologías que permiten obtener una visión del mundo real mezclada con elementos virtuales, lo que conlleva a una realidad que podríamos determinar mixta. Esta es la diferencia entre la realidad aumentada y la realidad virtual [12], pues mientras la primera como bien se ha dicho, combina los elementos reales y virtuales, la segunda sustituye completamente la realidad física por una virtual.

Una de las definiciones más concretas que se han dado sobre la realidad aumentada, fue la acuñada por Ronald Azuma [13] en 1997, quien la define con tres características básicas:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- Está registrada en 3D.

Aunque el término realidad aumentada fue creado en 1992, el avance de la tecnología en los ordenadores, consolas y dispositivos móviles inteligentes, ha permitido que cada vez sea más fácil crear aplicaciones de estas características y que éstas sean portátiles.

Podríamos describir una serie de elementos esenciales para la composición de un sistema de realidad aumentada:

- Elemento captor de las imágenes reales.
- Procesador que mezcla la información real con la creada virtualmente.
- Proyector de las imágenes tanto reales como virtuales. Normalmente es el mismo que el elemento captor (ordenadores, consolas, teléfonos móviles…).

Actualmente, la mayoría de los sistemas de realidad aumentada hacen uso de tecnologías como el GPS, los sensores o las brújulas, que permiten dotar a las aplicaciones de múltiples funcionalidades.

2.2. Historia de la Realidad Aumentada

La primera vez que se habló de realidad aumentada como tal fue en 1992, y el término fue acuñado por Tom Claudell. Ese mismo año, Feiner, McIntyre y Seligmann, crean un prototipo que por primera vez utiliza la realidad aumentada, se trata de *Karma[14]*, sistema que consiste en una pantalla colocada sobre la cabeza de los usuarios que mezcla la imagen real de una impresora láser (que el usuario debe tener delante) con imágenes virtuales que van mostrando cual debe ser el mantenimiento de dicha impresora.

Imagen 01. Sistema Karma

En 1998 tiene lugar por primera vez el congreso internacional sobre realidad aumentada 'International Workshop on Augmented Reality' [15], el cual sigue celebrándose cada año, actualmente bajo el nombre de 'International Symposium on Mixed and Augmented Reality'. En el año 2000, Bruce Thomas, junto con un grupo de estudiantes de una universidad de Australia, desarrolla *ARQuake* [16], el primer videojuego que utiliza este sistema para su uso en exteriores. Haciendo uso de un GPS, un ordenador portátil a la espalda y un casco con pantalla incorporada, los usuarios podían andar por la universidad visualizando el campus y matando a monstruos que 'aparecían'.

Imagen 02. Videojuego ARQuake

En 2008 aprovechando el auge de los teléfono inteligentes, sale a la venta la aplicación para Android *Wikitude AR Guía* [17], la cual proporciona información del

entorno sobre la imagen que capta la cámara del móvil. En 2009 se crea el logotipo oficial [18] de Realidad Aumentada, para estandarizar la aplicación de esta tecnología.

Imagen 03. Logotipo oficial de la Realidad Aumentada

2.3. Aplicaciones de la Realidad Aumentada

Antes de centrarnos en la realidad aumentada en los videojuegos, veamos que otras importantes aplicaciones pueden hacer uso de este sistema.

Educación

Una de las aplicaciones más interesantes de la realidad aumentada podría ser el campo de la enseñanza. Aquí podemos destacar a la empresa alemana Metaio[19], quienes sacaron en 2008 libros educativos basados en esta tecnología. Apuntando con ellos a la cámara web de un ordenador, se superponen objetos en 3D sobre el libro, mezclando en la pantalla ambas cosas.

Imagen 04. Libros educativos de Metaio.

Medicina

Combinando los equipos de imagen (escáner, ecografía, etc.) con sistemas informáticos, se han conseguido obtener representaciones de los órganos del cuerpo humano de una persona tan exactos que se han conseguido avances nunca conocidos en la medicina. Uno de los usos más prácticos de la realidad aumentada en medicina, es el campo de la cirugía, ya que para operaciones de alto riesgo, el cirujano dispone de una vista en tres dimensiones del órgano o los huesos que está interviniendo. Esto permite aumentar la seguridad, reduciendo los riesgos y los costes. Otra especialidad donde está presente cada vez más esta tecnología, son los ejercicios de rehabilitación, como ejercicios para personas con problemas psicomotrices [20].

Imagen 05.
Aplicación RA para problemas psicomotrices

Publicidad y venta

La publicidad utiliza todos los medios posibles para captar la atención de los usuarios, por lo que la realidad aumentada está entrando de manera potente en este negocio. La primera campaña de realidad aumentada en nuestro país vino de la mano de Fiat[21], quien para publicitar uno de sus coches, daba la oportunidad al usuario de crear su propio anuncio a través de la webcam de su ordenador. Por su parte, el mundo de la moda tampoco no se queda atrás y páginas como Zugara[22] o Tissot[23], permiten al usuario probarse sus prendas de ropa y relojes, respectivamente, sin salir de casa, haciendo uso de la cámara del ordenador.

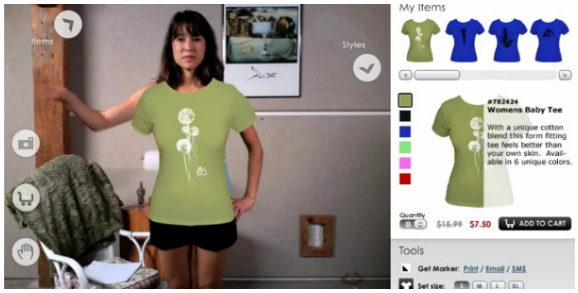


Imagen 06. Página web Zugara

Imagen 07. Página web Tissot

Turismo

Si quieres hacer turismo y tienes un teléfono de última generación, Wikitude [24] te muestra sobre la pantalla de éste la información sobre edificios, montañas, accidentes geográficos etc, a las que apuntas con la cámara del móvil. Haciendo uso del GPS, la brújula, y la cámara, se obtiene la información de donde se encuentra el usuario y se proporcionan los datos disponibles.

Imagen 08. Wikitude

Montaje y mantenimiento

Al igual que el prototipo 'Karma', la realidad aumentada cada día está más presente en equipos de montaje y mantenimiento. Formar a operarios inexpertos, reducir errores en cadenas de montaje o poner disminuir el tiempo empleado de un usuario para arreglar un aparato, son algunas de las ventajas que proporciona la realidad aumentada en este sector. Un ejemplo práctico sería una especie de guía que indica a un usuario como reparar un vehículo, indicándole en todo momento los pasos que debe ir siguiendo [25].

Imagen 09. RA para el montaje de un vehículo

Simulación

Un campo que se ha desarrollado de forma increíble gracias a la realidad aumentada ha sido el de las simulaciones [26]. Estos equipos que en sus inicios eran dispositivos analógicos caros, complicados y con pocas funciones, han pasado a ser elementos imprescindibles en formación y entrenamiento de muchos profesionales: pilotos, médicos, policías, conductores…

Muchas son las aplicaciones que hoy en día se le están dando a la realidad aumentada, el ejército, los proyectos industriales, los servicios públicos… son sólo algunos ejemplos. Hasta donde llegue la imaginación, llegará la realidad aumentada, y con el avance de la tecnología cada vez estará más presente en la vida diaria.

2.4. Realidad Aumentada en los videojuegos

Para comenzar hablar sobre los videojuegos, debemos remontarnos hasta 1958, cuando William Nighinbottham, utilizando un osciloscopio creó el juego Tennis for two [27] (Tenis para dos), el cual fue comercializado por Atari[28] en 1972 con el nombre de Pong. Ese mismo año, y tras varios intentos fallidos, Ralph Baer consigue comercializar la primera videoconsola, Magnavox Odyssey [29]. Comenzaba así una industria que evolucionaría a pasos agigantados.

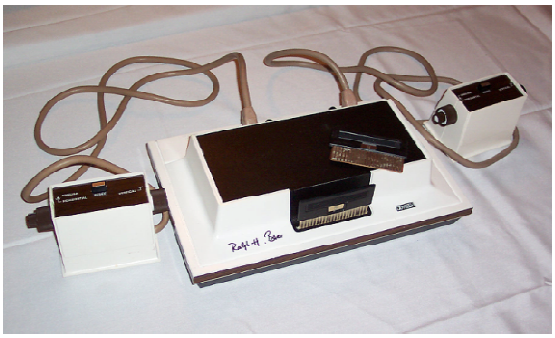


Imagen 10. Magnavox Odyssey

Antes de proseguir con la evolución de los videojuegos, es curioso destacar como las grandes empresas comercializadoras de éstos nacieron mucho antes que los videojuegos y sus actividades eran diversas. Nintendo [30] fue la primera de ellas, fundada en 1889, y dedicada a la fabricación de barajas de cartas. En 1946 le tocó el turno a Sony [31], quien apareció como reparadora de equipos electrónicos. Taito nace en 1963 y en sus inicio distribuía máquinas expendedoras. Sega[32], fundada en 1965, vendía gramolas y Konami[33], creada en 1969, se dedicaba a la reparación de las mismas.

Con el éxito de Pong, Atari comercializó rápidamente otros videojuegos como Space Race, Pong Doubles y Gotcha. Ninguno alcanza el éxito conseguido con Pong, y Atari se "pone las pilas" para sacar al mercado una videoconsola que pudiera ser utilizada en los televisores de casa con el juego del Pong [34]. Lo consigue en 1974, y dicha consola lleva el nombre de Home Pong [35]. A partir de este momento, las empresas ven un negocio prometedor en el sector de los videojuegos y las videoconsolas y Nintendo, en 1977, lanza su primera videoconsola: Color TV Game 6. En esta misma década cabe destacar el Space Invaders, de Taito, y el Pac Man de Namco, este último considerado, junto con el Pong, el videojuego más influyente de la historia.

. Home Pong

La década de los 80 [36] destacaría por la gran crisis en la industria y por la creación de juegos y videoconsolas portátiles que supondrían una gran revolución, como el Super Mario [37] y la Game Boy [38].

En 1980 Nintendo comienza a crecer imparablemente y saca a la venta la consola portátil Game & Watch [39], la cual tenía un solo juego y a la vez era reloj y alarma. En 1981 saca a la venta el juego Donkey Kong donde aparece por primera vez el archi-famoso personaje de Mario.

En 1982, salen al mercado varias consolas como Mattel Intellivision, Colecovision o Atari 5200, y ordenadores como el Commodore 64 [40] y ZX Spectrum. Pero este año sería también el predecesor de una gran crisis. El mercado se encuentra plagado de videojuegos y videoconsolas, unas clones de otras, y a las que no se puede dar salida, por lo que la industria atraviesa una grave crisis. Y no será hasta mediados de la década cuando empiece a recuperarse. Más tarde Nintendo saca el juego de Mario Bros [41], lo que supuso una auténtica revolución, llegando a vender más de 10 millones de copias. También es el turno de juegos míticos como el Tetris [42] o Boberman [43] y de videoconsolas como la Master System [44]. La auténtica revolución en las consolas llegaría en 1989 con el nacimiento de la Game Boy, la videoconsola portátil más famosa de la historia.

Imagen 12. Game & Watch

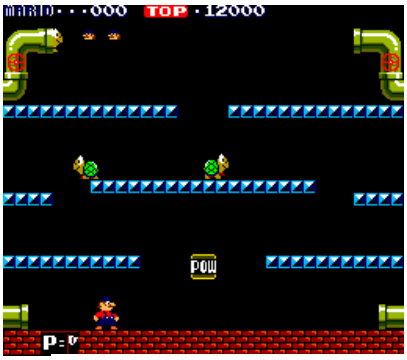


Imagen 13. Mario Bros

En 1990 [45] sale a la venta la consola Super NES, conocida más popularmente como Super Nintendo [46], la cual gracias a juegos como Super Mario Kart, Donkey Kong Country o Yoshi's Island arrasó en todo el mundo. Comenzaba así una gran rivalidad entre ella y la Mega Drive [47] de Sega (implantada ya en Estados Unidos). Este mismo año Sega saca a la venta Game Gear, para hacer la competencia a la Game Boy, y a pesar de su gran tamaño para tratarse de una consola portátil y de la escasa duración de su batería, tuvo una buena aceptación por el público. Al año siguiente Sega saca a la venta el juego Sonic [48], para hacer frente a Super Mario. Sonic vendió

más de cuatro millones de copias en todo mundo, diferenciándose de Super Mario por tener mayor velocidad y dinamismo en el juego.

También en 1991 saldría a la venta la segunda versión de Street Fighter [49], que dada su popularidad fue adaptado para la mayoría de las videoconsolas domésticas. En 1992 sale a la venta Wolfenstein 3D, considerado como el primer videojuego que aprovecha con éxito la perspectiva tridimensional.

Imagen 14. Wolfenstein 3D

El 1993 aparece el primer videojuego de la saga FIFA [50], el FIFA International Soccer, que se lanzó en muchísimos soportes y arrasó en ventas. En 1994 Sony saca a la venta la PlayStation [51], que se convirtió en una de las más vendidas de la historia, destacando juegos como Metal Gear Solid, Resident Evil, Tomb Raider, Tekken, Gran Turismo, ISS Pro Evolution, Final Fantasy o Crash Bandicoot. Ese mismo año aparece el superventas Warcraft, que supuso un punto de inflexión en los juegos de estrategia en tiempo real. En 1995 Mario Bros se une a la perspectiva 3D con su videojuego Super Mario 64, del cual se vendieron más de doce millones de copias.

Imagen 15. Super Mario 64

En 1997 aparecen grandes videojuegos como Final Fantasy VII [52], considerado por muchos una película hecha videojuego, debido a la gran cantidad de secuencias de video que incluía. Grand Theft Auto [53] se hace un hueco en el mercado a pesar de sus gráficos bastante pobres. Este año, de la mano del juego Snakes, comienza el mercado de los videojuegos para teléfonos móviles [54]. En 1998 cabe destacar el juego Gran Turismo, el juego más vendido de la PlayStation, y que fue el primero de una saga que en 2006 apareció en HD.

En el año 2000 [55], sale al mercado la PlayStation2, sin duda la mejor consola de la historia, al menos hasta el momento, la cual no tuvo rival en el mercado y consolas como Game Cube o Xbox, quedarían siempre en un segundo plano. Ese mismo año aparece el juego para DreamCast, Shenmue [56], que se definió como FREE, Reactive Eyes Entertainment (Entretenimiento Visual de Reacción completa), queriendo con él expresar los movimientos y acciones que dispone el juego, y su gran interactibilidad, como por ejemplo poder llamar a los timbres de las casas que aparecen, hablar con todos los personajes del juego, comprar en las tiendas, etc.

Imagen 16. Shenmue

A partir del año 2003 aparecen infinidad de consolas portátiles, revolucionando el mercado en 2004 la Nintendo DS, con la inclusión de una pantalla táctil, un micrófono con reconocimiento de voz, conexión wifi [57] que permitía jugar partidas online, etc. El amplio mercado de estas videoconsolas puede considerarse el inicio del sector de los videojuegos para móviles, poder disfrutar de grandes juegos con un pequeño aparato y en cualquier sitio. En 2006 sale a la venta la PlayStation3 [58] la cual no tardará en unirse al mercado de los videojuegos de realidad aumentada. Más tarde, en 2010 aparece Nintendo 3DS [59], con la capacidad para mostrar 3D real sin necesidad de ningún tipo de lente.

En esta evolución de videoconsolas y videojuegos también han tenido su espacio los juegos que utilizan realidad aumentada. Junto con el ya mencionado ARQuake, apareció también en el año 2000, implementado por la universidad de Singapur, el Pac-Man para realidad aumentada [60]. El usuario podía tomar el rol de un fantasma o del propio Pac-Man, y el laberinto del juego eran las calles de Singapur. Para poder jugar era necesario un ordenador portátil, unas gafas, GPS, bluethooth, wifi sensores e infrarrojos.

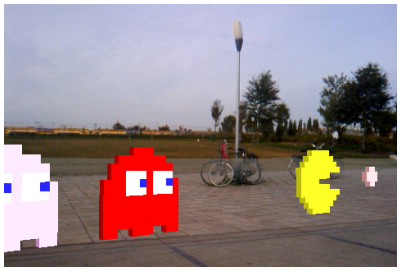


Imagen 17. PacMan

Las videoconsolas tampoco se quedaron atrás y poco a poco se han ido introduciendo en el mundo de la realidad aumentada. Nintendo Dsi lanzó a finales de 2010 el juego Ghostwire [61], en el que se superponen fantasmas sobre la cámara del dispositivo, y el usuario con ayuda de la pantalla táctil y el micrófono, debe resolver puzles para que estos fantasmas desaparezcan. Por su parte Sony sacó el juego Invizimals [62] para la PSP. El usuario, en un entorno real a través de la cámara del dispositivo, debe capturar a unas pequeñas criaturas llamadas invizimals, para luego combatir con ellas contra otras controladas por el juego o por otros usuarios con PSP.

Imagen 18. Ghostwire

Imagen 19. Invizimals

Microsoft [63] creó Kinect para la Xbox 360, la cual permite a los usuarios jugar sin utilizar ningún mando, sino su propio cuerpo. Esto permite disfrutar de juegos tan clásicos como el Tetris, utilizando realidad aumentada. La consola más vendida de todos los tiempos tampoco se queda atrás y mediante la webcam (PlayStation Eye) podemos disfrutar de grandes juegos de realidad aumentada como el famoso EyePet [64], que permite al usuario interaccionar con una mascota virtual realizando actividades como si realmente estuviera en nuestra casa.

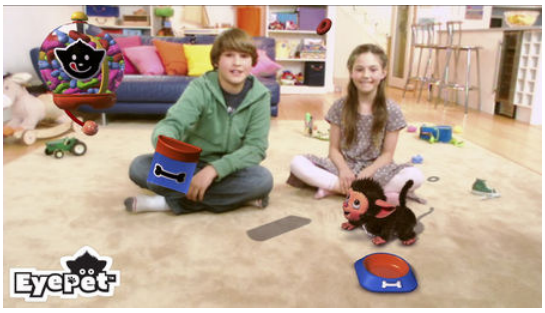


Imagen 20. EyePet

Sin embargo el desarrollo de la tecnología que ha permitido que en un teléfono móvil podamos contar con cámara de vídeo, GPS, sensores, internet… ha convertido a estos dispositivos en grandes portadores de aplicaciones con realidad aumentada. Entre las diferentes aplicaciones, que utilizan este sistema, podemos destacar:

- Layar: con ayuda del GPS para obtener la localización del usuario, superpone capas a la imagen de la cámara del móvil para mostrar información de lo que tenemos alrededor, como restaurantes, lugares de ocio, etc.

- TwittAround [65]: permite al usuario ver en tiempo real todos los tweets que se están publicando cerca a su ubicación.

- Wikitude World Browser [66]: concebida como una enciclopedia del futuro, presenta al usuario información de su entorno, de una manera mucho más limpia y ordenada que en Layar, aunque cuenta con menor cantidad de capas. Está incluida entre una de las 50 mejores aplicaciones para dispositivos Android.

- TAT augmented ID: sin duda la aplicación de realidad aumentada más polémica hasta el momento. Realizando un vídeo sobre una persona, se muestran todos los datos que hay sobre ésta en la red.

- Yelp Monocle: es una aplicación creada para ser utilizada en Estados Unidos y que muestra al usuario información sobre locales comerciales, especialmente sobre restaurantes, cercanos a la ubicación del usuario. Ofrece información como calidad, otorgada por otros usuarios, descripción, etc. Sin duda una buena manera de descubrir los mejores restaurantes en ciudades como Nueva York o Los Ángeles.

Imagen 21. Wikitude World Browser

Imagen 22. TweetAround

Centrándonos por fin en los videojuegos para dispositivos móviles que utilizan realidad aumentada, hay que decir que no son muy populares y la mayoría de la gente no los conoce. Posiblemente sea porque en la mayoría de los casos hace falta disponer de un potente teléfono móvil, con conexión a internet y dispositivo GPS, y aunque cada vez es más la gente que va comprando este tipo de dispositivos, se puede decir que están empezando a conocerse por el público en general. Por ello se puede esperar que con los años cada vez sean más los videojuegos que se basen en la realidad aumentada y que éstos sean conocidos por gran parte de las personas aficionadas a los videojuegos.

Uno de los primeros videojuegos que apareció en este campo fue ARhrrrr [67], creado por la universidad de Georgia Tech junto con la empresa Nvidia, una de las grandes líderes del sector de tecnologías de visualización digital. El usuario apuntando con su cámara que simula un helicóptero que sobrevuela una cuidad, debe defender dicha ciudad que es atacada por zombis antes de que éstos maten a los civiles. El terminal debe apuntar hacia un mapa, que provocará que vayan apareciendo los zombis y los edificios 3D irán también apareciendo en el terminal según hacia que parte del mapa apuntemos, como se muestra en la siguiente fotografía:

Imagen 23. ARhrrrr

La empresa int13 desarrolló un juego al estilo de los míticos Pokémon, Kweekies [68], donde dos especies de animales luchan por turnos en un combate. El usuario es el entrenador del animal. Hace falta, además del dispositivo, un código QR.

Imagen 24. Kweekies

En la introducción de esta memoria, se hacía referencia al hecho de que iPhone y Android abarcan casi todo el mercado de los teléfonos de última generación a día de hoy. Y como no podía ser menos, ambos poseen el mayor catálogo de videojuegos que utilizan la realidad aumentada.

En iPhone podemos destacar juegos como Shadow Cities [69], donde el escenario del juego es la ciudad donde se encuentra el usuario, quien toma el papel de un mago que tiene que evitar que la magia desaparezca del mundo. Se presenta al usuario un mapa de la zona donde se encuentra y su ubicación específica en él, por lo que es necesario disponer de GPS en el dispositivo. Sobre el mapa aparece un fuego,

indicando el lugar que hay que conquistar. Sin embargo, se puede considerar como pega que es un juego multijugador, y según el número de personas que estén jugando, podremos hacer más o menos cosas. En sus inicios, solo estuvo disponible en Finlandia, donde tuvo bastante éxito, pero en nuestro país, no hay muchos jugadores, por lo que resulta bastante aburrido jugar. Los usuarios tienen que completar misiones, luchar entre ellos por la conquista de barrios y ciudades y ganar energía y poderes para convertirse en el mago más fuerte. Aunque la instalación del juego es gratuita, para poder aumentar de nivel más rápidamente y conseguir la energía necesaria, hay que pagar.

En Gigaputt [70] se puede jugar al golf en la ciudad donde te encuentres, la cual se convierte en un verdadero campo de golf. También es necesario disponer de GPS en el dispositivo, con el que se determina la posición del usuario y se fijan tres hoyos alrededor de él. El golpeo de la pelota se realiza con el teléfono a modo de palo de golf. Permite jugar hasta un máximo de tres usuarios en el mismo dispositivo.

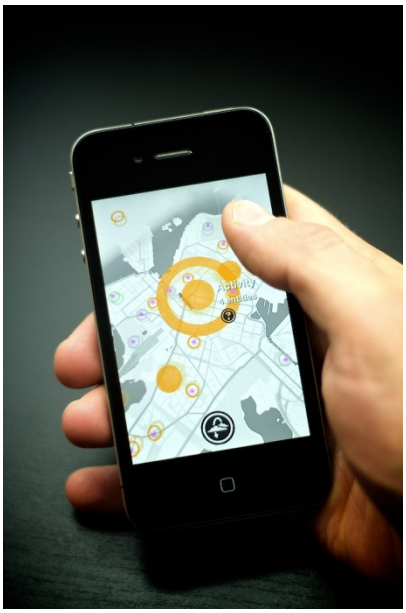


Imagen 25. Shadow Cities

Imagen 26. Gigaputt

Por su parte ARDefender [71], disponible tanto para iPhone como para terminales Samsung basados en el sistema Bada [72], fue creado por la empresa int13. Antes de comenzar a jugar hay que descargar e imprimir una plantilla, y colocando el objetivo del dispositivo hacia él, aparece una torre que es el eje central del videojuego. A la imagen capturada por la cámara se le van añadiendo los objetos 3D correspondientes, y el usuario debe tratar de eliminar a los enemigos que aparecen con las diferentes armas que proporciona el juego.

Imagen 27. ARDefender

Por otro lado, en el Market de Android, podemos encontrar videojuegos que utilizan esta tecnología, tales como SpecTrek [73], otro entretenido juego de realidad aumentada. Games4all se ha encargado de crear un videojuego donde puedes sentirte como un verdadero caza-fantasmas. Ese es el objetivo del juego, cazar a los fantasmas que SpecTrek ha colocado estrategicamente a tu alrededor. Si lo que buscas es salir de casa y hacer ejercicio de forma entretenida, este juego te ofrece la posibilidad. Además te ofrece la opción de jugar con más o menos duración, ya que existen tres modalidades. Hace uso del GPS del dispositivo para localizar la posición del usuario y de los fantasmas escondidos. Se puede descargar gratis de Android Market.

Imagen 28. SPecTrek

Zombie, Run [74], como bien dice el nombre de éste, Zombies, ¡corre!, de eso trata. Imagina que vives en un lugar infectado de hordas y hordas de zombies, no te queda otra que correr. Pues bien, este juego te permite saber cuál es su ubicación y de esta manera poder esquivarlos. Cuanto más tiempo logres esquivarlos, mayor puntuación lograrás. A medida que vayas cogiendo práctica, puedes armarte de valentía y subir el nivel del juego, es decir, mayor número y velocidad de los Zombies. Una forma entretenida de quemar calorías. Se puede descargar gratis de Android Market.

Imagen 29. Zombie, Run

Precisión, puntería, calma y buenos reflejos. Con estas premisas podrás ser un buen jugador de Sky Siege [75], un juego donde podrás disparar desde tu avión a todo enemigo que se cruce por tu mirilla. Solo hace falta que enciendas la cámara de tu terminal para que aparezca ésta. Desde tu propia casa podrás creerte que multitud de aviones y helicópteros te atacan e intentan invadir tu territorio. Además, conforme pase el tiempo, el número de aviones y helicópteros que te irán acosando será cada vez mayor, dificultando de esta manera tu labor (existen 32 niveles de dificultad). Pero eso sí, siempre podrás utilizar distinto tipos de armas dependiendo del momento y el enemigo con el que te topes. Se puede descargar de Android Market por 1,69€.

Imagen 30. Sky Siege

Parallel Kingdom – Age of Emergence [76]. Uno de los juegos más famosos y divertidos dentro de la realidad aumentada. Gracias a este videojuego podrás sumergirte en un mundo totalmente sorprenderte, donde convivirás con multitud de monstruos y otros jugadores. Dentro de este mundo virtual podrás mantener conversaciones a tiempo real con usuarios que también estén jugando en ese momento ó matar a los monstruos y a los jugadores que consideres oportuno con distintos tipos de armas, las cuales podrás ir adquiriendo y mejorando conforme vayas jugando. Estas y otras muchas funciones más conforman Parallel Kingdom. Se puede descargar gratis de Android Market.

Capítulo 3.

Desarrollo

3.1. Análisis

3.1.1. Casos de uso

El primer punto del análisis va a consistir en la identificación de los casos de uso (CU). Un caso de uso es una secuencia de acciones que se dan entre un sistema y sus actores como respuesta a un evento que inicia un actor sobre dicho sistema.

En primer lugar se presenta el diagrama de casos de uso, el cual muestra de una manera sencilla los límites del sistema y cómo se utiliza. Sirve como una herramienta de comunicación que resume el comportamiento de un sistema y de sus actores (en nuestro caso solo tenemos un actor, el jugador). Posteriormente se analizan los diferentes casos de uso, especificando en cada caso los actores, las precondiciones, las postcondiciones, el escenario principal, y los escenarios alternativos y de excepción cuando sean precisos.

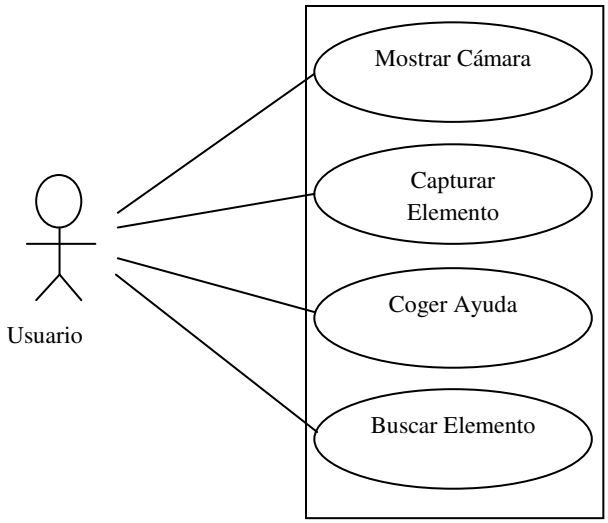


Imagen 31. Diagrama de casos de uso

CU01: Mostrar Cámara

Actores

- Usuario.

Precondiciones

- Haber iniciado el sistema.

Postcondiciones

- Se muestra la imagen de la cámara.

Escenario Principal:

1. El usuario inicia el sistema de realidad aumentada.
2. El sistema muestra la imagen capturada por la cámara.

CU02: Capturar Elemento

Actores

- Usuario.

Precondiciones

- Haber encontrado un elemento a través de la pantalla del dispositivo móvil.

Postcondiciones

- Hay un elemento menos a capturar.

Escenario Principal:

1. El sistema muestra un aviso de que es posible capturar un elemento.
2. El usuario selecciona la imagen del elemento sobre la pantalla.
3. El sistema muestra un aviso al usuario de que el elemento ha sido capturado.
4. El sistema muestra un efecto sonoro.

Escenarios Alternativos:

2a. Usuario fuera del alcance

1. El usuario modifica su localización de forma que queda fuera del alcance.
2. El caso de uso finaliza.

*b. El usuario puede seleccionar la opción “Atrás”

1. El usuario selecciona el botón Atrás.
2. EL sistema finaliza.
3. El caso de uso finaliza.

CU03: Coger Ayuda

Actores

- Usuario.

Precondiciones

- Haber encontrado una ayuda a través de la pantalla del dispositivo móvil.

Postcondiciones

- El sistema efectúa las características de la ayuda conseguida.

Escenario Principal:

1. El sistema muestra un aviso de que es posible coger una ayuda.
2. El usuario selecciona la imagen del ítem sobre la pantalla.
3. El sistema muestra un aviso al usuario de que el ítem ha sido cogido.
4. El sistema muestra un efecto sonoro.

Escenarios Alternativos:

2a. Usuario fuera del alcance

1. El usuario modifica su localización de forma que queda fuera del alcance.
2. El caso de uso finaliza.

*b. El usuario puede seleccionar la opción “Atrás”

1. El usuario selecciona el botón Atrás.
2. EL sistema finaliza.
3. El caso de uso finaliza.

CU04: Buscar Elemento

Actores

- Usuario.

Precondiciones

- Haber iniciado el sistema.

Postcondiciones

- Poder capturar el elemento encontrado.

Escenario Principal:

1. El usuario modifica la posición del móvil buscando elementos.
2. El sistema verifica si el usuario se encuentra en el rango de visión de algún elemento.
3. El sistema verifica si el usuario se encuentra en el rango de visión de alguna ayuda.
4. El sistema visualiza el elemento.
5. El sistema visualiza la ayuda.

Escenarios Alternativos:

3a. No se visualiza objeto
1. El usuario no se encuentra dentro del rango de visualización de ningún elemento.
2. Volver al paso 1.

5a. No se visualiza objeto
1. El usuario no se encuentra dentro del rango de visualización de ningún elemento.
2. Volver al paso 1.

*b. El usuario puede seleccionar la opción "Atrás"
1. El usuario selecciona el botón Atrás.
2. EL sistema finaliza.
3. El caso de uso finaliza.

3.1.2. Identificación de requisitos

El siguiente punto del análisis va a consistir en la identificación de los requisitos de usuario y los requisitos software, los cuales determinarán las necesidades y condiciones a satisfacer en la creación del sistema.

Mediante los requisitos de usuario se puede conocer lo que el usuario final espera del sistema y así poder cubrir todas sus necesidades. Los requisitos detallados a continuación vienen descritos por las siguientes características:

- Identificador: cadena de caracteres que identifica de manera única a cada requisito.

- Nombre: título del requisito.

- Origen: entidad que origina el requisito.

- Verificable: especifica si el cumplimiento del requisito es verificable con una prueba. Tendrá el valor "Si" en caso de que sea verificable, y "No" en caso de que no lo sea.

- Necesidad: indica si el requisito es esencial para el proyecto o no. Tendrá en valor "Esencial" si lo es, y "Opcional" en caso contrario.

- Claridad: indica si el requisito se entiende bien o es ambiguo. Tendrá en valor "Si" en caso de que se entienda bien, y "No" en caso de que sea ambiguo.

- Prioridad: indica la prioridad del requisito mediante la escala "Alta", "Media" y "Baja".

- Estabilidad: indica si el requisito puede cambiar a lo largo del proyecto. Se mide con una escala de tres valores “Alta” (estabilidad máxima) “Media” (estabilidad media) y “Baja” (estabilidad mínima).

RU 01 - Imagen de fondo			
Descripción	Mostrar de imagen de fondo la imagen capturada por la cámara del dispositivo.		
Origen	CU01	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RU 02 - Visualizar elementos a capturar			
Descripción	El usuario podrá visualizar sobre la imagen de la cámara los elementos disponibles.		
Origen	CU04	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RU 03 - Visualizar ayudas			
Descripción	Si se hace de esta opción, el usuario podrá visualizar sobre la imagen de la cámara las ayudas que puede coger.		
Origen	CU04	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Media

RU 04 - Capturar elementos/ayudas			
Descripción	El usuario podrá capturar los elementos y las ayudas mediante la pantalla táctil del móvil.		
Origen	CU02, CU03	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RU 05 - Visualizar elementos según geolocalización			
Descripción	El usuario podrá visualizar sobre la imagen de la cámara los elementos según la geolocalización de éstos.		
Origen	CU04	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RU 06 - Cambiar tamaño según geolocalización			
Descripción	El usuario podrá visualizar sobre la imagen de la cámara los elementos en un tamaño diferente según la geolocalización de éstos.		
Origen	CU04	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Media

RU 07 - Soportar transparencias			
Descripción	Los elementos creados deberán soportar transparencias para poder visualizarse sobre la imagen de la cámara.		
Origen	CU04	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RU 08 - Dispositivos Android			
Descripción	El sistema deberá ejecutarse sobre dispositivos android con cámara y utilizar los recursos de forma eficiente.		
Origen	CU01, CU02	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

Dentro de los requisitos software debemos diferenciar entre requisitos funcionales y requisitos no funcionales.

Los requisitos funcionales describen las funciones que se esperan del sistema. Por su parte, los requisitos no funcionales, son restricciones sobre los requisitos funcionales. Ambos van a estar caracterizados por las mismas propiedades que los requisitos de usuario.

A continuación se detallan los requisitos funcionales de nuestro sistema:

RSF 01 - Imagen de la cámara como fondo			
Descripción	El sistema debe mostrar de imagen de fondo la imagen capturada por la cámara del dispositivo.		
Origen	RU 01	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSF 02 - Pintar elementos/ayudas según posición			
Descripción	Los elementos y las ayudas, generados en 3D, se pintarán de acuerdo a la orientación del dispositivo móvil.		
Origen	RU 02, RU 03, RU05	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSF 03 - Visualizar según distancia			
Descripción	Los elementos y las ayudas sólo serán visibles si se encuentran a menos de 20 metros de la posición del usuario.		
Origen	RU 02, RU 03, RU05	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Media

RSF 04 - Capturar			
Descripción	Los elementos y las ayudas sólo se podrán capturar si se encuentran a menos de 7 metros de la posición del usuario.		
Origen	RU 04	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Media

RSF 05 - Visualización según dirección			
Descripción	Los elementos y las ayudas sólo se podrán visualizar si se encuentran en la dirección en la que apunta el móvil y dentro del ángulo que recoge la cámara.		
Origen	RU 02, RU 03, RU05	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSF 06 - Pintar según distancia			
Descripción	Los elementos y las ayudas se pintaran en tamaño y posición en función de la distancia a la que se encuentren.		
Origen	RU 02, RU 03, RU06	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Alta

RSF 07 - Aviso capturar			
Descripción	El sistema mostrará un aviso cuando los elementos/ayudas se encuentren a menos de 7 metros indicando que pueden ser capturados.		
Origen	RU 04	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Alta

RSF 08 - Aviso visual			
Descripción	El sistema mostrará un aviso visual cuando se haya capturado un elemento o una ayuda.		
Origen	RU 04	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Media	**Estabilidad**	Alta

RSF 09 - Aviso sonoro			
Descripción	El sistema mostrará un aviso sonoro cuando se haya capturado un elemento o una ayuda.		
Origen	RU 04	**Verificable**	Si
Necesidad	Opcional	**Claridad**	Si
Prioridad	Baja	**Estabilidad**	Alta

RSF 10 - Mostrar todos los elementos y ayudas			
Descripción	El sistema debe mostrar todos los elementos y ayudas como sea necesario según la posición en la que se encuentren.		
Origen	RU 02, RU 03	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSF 11 - Transparencias			
Descripción	Los elementos creados deberán soportar transparencias para poder visualizarse sobre la imagen de la cámara.		
Origen	RU 07	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

Por su parte, los requisitos no funcionales serán:

RSNF 01 - Aplicaciones móviles android			
Descripción	El sistema solo puede integrarse en aplicaciones móviles android.		
Origen	RU08	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 02 - Versión mínima 2.2			
Descripción	El sistema está diseñado para una versión mínima 2.2		
Origen	RU08	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 03 - Dispositivo con cámara			
Descripción	El sistema solo puede utilizarse en dispositivos con cámara de video.		
Origen	RU 01	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 04 - Dispositivo con pantalla táctil			
Descripción	El sistema solo puede utilizarse en dispositivos con pantalla táctil.		
Origen	RU 04	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 05 - Formatos de imágenes y audio			
Descripción	Las imágenes utilizadas serán del formato .jpg, .bmp o gif, y los efectos de sonido en formato .wav		
Origen	RU02, RU03	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 06 - Restricciones de batería			
Descripción	El sistema debe hacer un uso mínimo de batería para poder ejecutarlo durante el tiempo deseado.		
Origen	RU08	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

RSNF 07 - Tiempo de respuesta			
Descripción	El tiempo de respuesta del sistema frente a una petición de usuario debe ser menor de 2 segundos.		
Origen	RU08	**Verificable**	Si
Necesidad	Esencial	**Claridad**	Si
Prioridad	Alta	**Estabilidad**	Alta

3.2. Diseño Conceptual

Para poder llevar a cabo el desarrollo del sistema, hay que saber qué módulos lo componen, que módulos componen las posibles aplicaciones en las que se integre, y cómo se relacionan éstos.

3.2.1. Arquitectura del sistema

En primer lugar se deben identificar los principales módulos en los que se descompone el presente sistema de realidad aumentada:

- *Sensores*, encargado del control de los sensores del dispositivo.
- *Gráficos*, para pintar los objetos mediante el estándar OpenGL.
- *Eventos de pantalla*, con el que se controlarán las pulsaciones táctiles sobre la pantalla del dispositivo.
- *Sonido*, encargado de proporcionar los diferentes efectos de sonido al sistema.

Para poder ser integrado en la creación de diversas aplicaciones, estos módulos deben integrarse con al menos otros tres:

- *Interfaz de usuario*, para poder iniciar la aplicación correspondiente, permitiendo al usuario, entre otras opciones, elegir si se desean utilizar o no los efectos de sonido disponibles.
- *Posicionamiento*, encargado de suministrar al módulo de los sensores, las correspondientes coordenadas geográficas con las que trabajar para determinar la posición del usuario y los objetos.
- *Lógica*, para controlar el número de objetos a pintar, la utilización de ítems, las consecuencias de los eventos táctiles…

Para entender mejor como interaccionan entre sí estos módulos para la creación de una aplicación, utilizaremos el ya citado videojuego *Invasión Androide*, el cual a parte de los tres módulos mínimos necesarios para la creación de cualquier aplicación que utilice el sistema de realidad aumentada incluye un cuarto módulo, *Persistencia*, encargado de dotar al videojuego de la posibilidad de almacenar los records conseguidos en las diferentes partidas que se realicen.

De esta manera, obtenemos el siguiente diagrama de interacción:

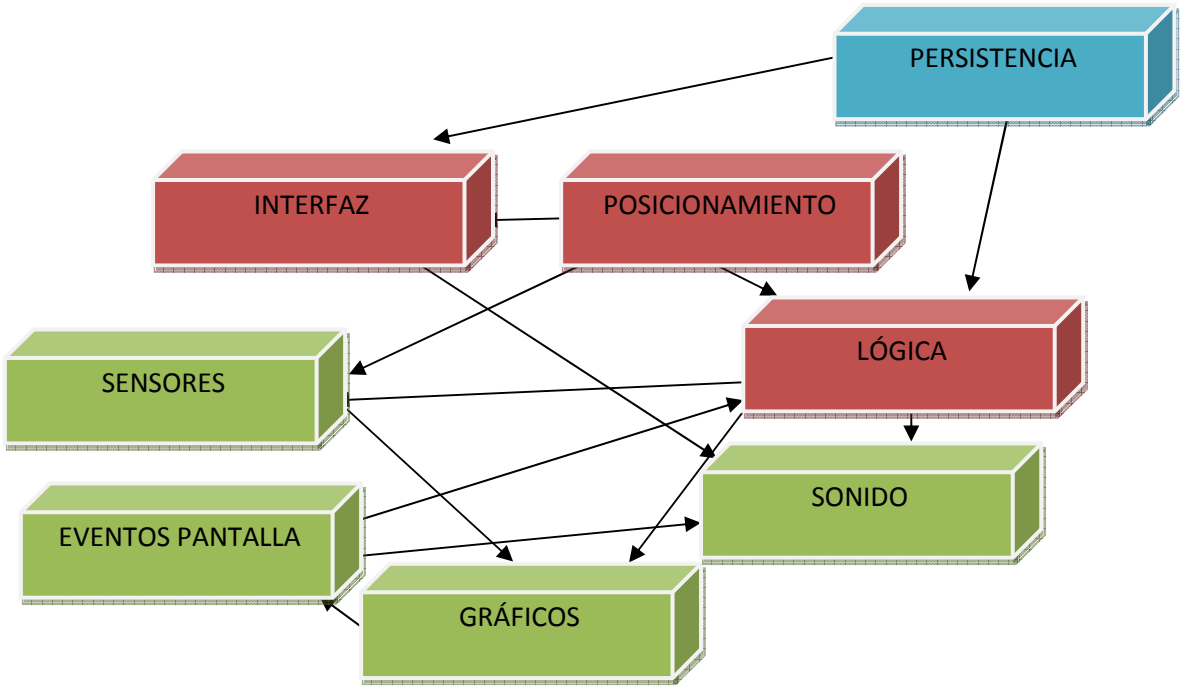


Imagen 32. Diagrama de interacción entre módulos, 1

- Módulos pertenecientes al sistema de realidad aumentada.
- Módulos mínimos a incluir para crear cualquier aplicación que utilice el presente sistema.
- Módulo adicional que incluye el videojuego *Invasión Androide*.

Una vez identificados los diferentes módulos, debe estudiarse como interaccionan entre sí cada uno de ellos, utilizando para la explicación la integración del sistema en el desarrollo del videojuego Invasión Androide.

Interacción 1

Imagen 33. Interacción módulos, 1

Cuando se inicie una nueva partida del videojuego, se mostrará un mapa de la zona en la que se encuentra el usuario, mostrando la posición de éste y de los robots a capturar. Por tanto, el módulo interfaz mostrará el mapa proporcionado por el módulo de posicionamiento.

Interacción 2

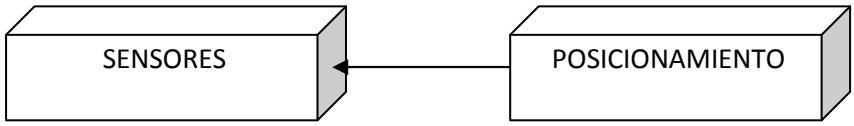


Imagen 34. Interacción módulos, 2

El módulo de sensores, entre otras, se encargará de calcular si el objetivo de la cámara del dispositivo apunta en la dirección en la que están los robots y la distancia a la que se encuentran para poder pintarlos. En necesario por ello que el módulo de posicionamiento le facilite las coordenadas GPS tanto del usuario como de los robots a capturar.

Interacción 3

Imagen 35. Interacción módulos, 3

El módulo de sensores, calculará la dirección en la que se encuentra el usuario de todos los robots a capturar en la ronda en la que está el videojuego, cuyas posiciones han sido generadas aleatoriamente por el módulo de lógica.

Interacción 4

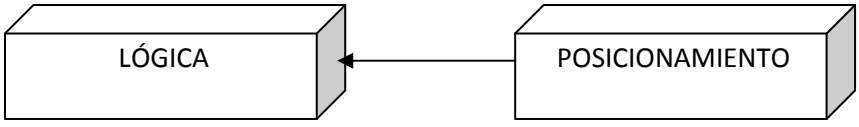


Imagen 36. Interacción módulos, 4

El módulo de lógica necesita de una serie de parámetros para desarrollar todas las funcionalidades que en este módulo se implementan. Como el posicionamiento de

los robots y los ítems, o el desplazamiento de los robots. Para ello recibe del módulo de posicionamiento los atributos de la ubicación del usuario, el rango de coordenadas que se puede usar para colocar marcadores y área limitada de actuación.

Interacción 5

Imagen 37. Interacción módulos, 5

El módulo de gráficos se encargará de pintar los robots y los ítems que procedan en cada momento. Para saber tanto el tamaño como la posición a la que pintar éstos, necesita que el módulo de sensores le proporcione la distancia y el ángulo en el que se encuentran respecto al usuario. De la misma manera, para poder girar y rotar dichos objetos en función del dispositivo móvil, es necesario obtener los valores de pitch y roll, explicados en el siguiente capítulo, proporcionados por el módulo de sensores.

Interacción 6

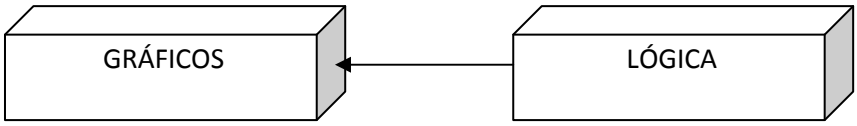


Imagen 38. Interacción módulos, 6

Otro requisito para poder pintar tanto los robots como los ítems, es que ambos sean visibles desde la posición del usuario, es decir, que no haya ningún obstáculo entre ambos que impida su visualización. Este dato es proporcionado por el módulo de lógica.

Interacción 7

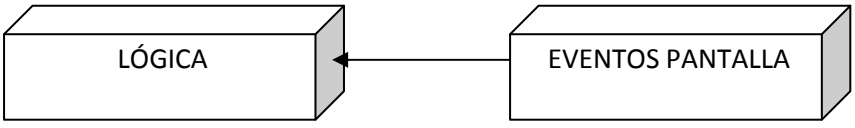


Imagen 39. Interacción módulos, 7

Para poder llevar a cabo las diferentes acciones que derivan de la captura de los robots o los ítems, tales como el paso a una ronda superior o disfrutar de las ayudas

ofrecidas por los diferentes ítems, el módulo de lógica necesita que el módulo que se encarga de los eventos de la pantalla, le facilite cuándo éstos han sido capturados.

Interacción 8

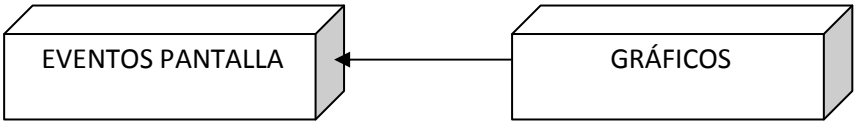


Imagen 40. Interacción módulos, 8

Para saber en qué posición de la pantalla están ubicados los robots o las ayudas a capturar, es necesario que el módulo de gráficos le facilite esta información al módulo de los eventos de pantalla.

Interacción 9

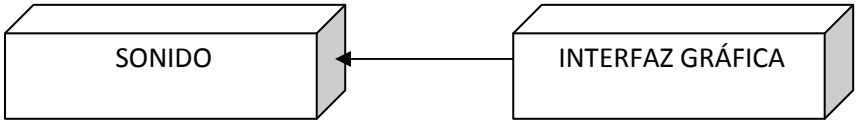


Imagen 41. Interacción módulos, 9

El módulo de sonido proporcionará efectos sonoros al videojuego siempre y cuando esta opción haya sido activada, o por lo menos no haya sido desactivada (ya que en caso de no elegir ninguna los sonidos se activan por defecto). Por ello necesita conocer el estado de activación o desactivación de los efectos, proporcionado por el módulo de interfaz gráfica.

Interacción 10

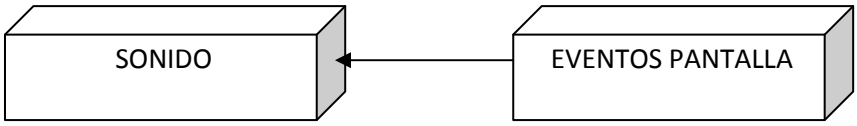


Imagen 42. Interacción módulos, 10

El módulo de sonido proporcionará efectos, siempre que éstos estén activados, cuando un robot o un ítem hayan sido capturados, por lo que necesitará esta información proporcionada por el módulo de eventos de pantalla.

Interacción 11

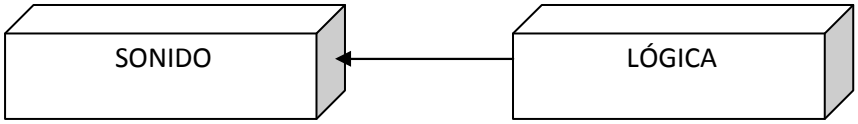


Imagen 43. Interacción módulos, 11

De la misma manera que en el caso anterior, el módulo de sonido proporcionará un efecto sonoro cuando el tiempo de la partida esté finalizando, información controlada por el módulo de lógica.

Interacción 12

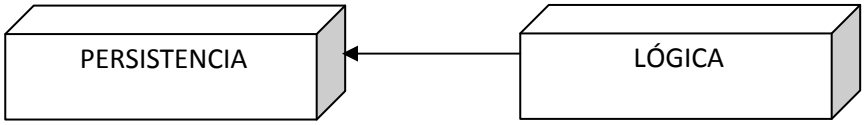


Imagen 44. Interacción módulos, 12

El módulo de lógica le facilitará al módulo de persistencia la puntuación obtenida por un usuario a lo largo de una partida. De esta manera se podrán almacenar los tres mejores resultados.

Interacción 13

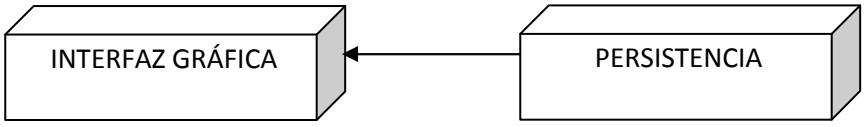


Imagen 45. Interacción módulos, 13

En este caso será el módulo de persistencia el que ofrezca los resultados de las tres mejores puntuaciones a la interfaz gráfica, para que éste pueda mostrárselos al usuario, si es que desea visualizarlos.

3.2.2. Descripción de los módulos

Una vez analizadas todas las interacciones entre módulos, pueden deducirse fácilmente cuáles de éstas se producen entre módulos de este sistema y cuáles entre módulos de este sistema y los del videojuego, sirviendo de referencia para cualquier otra aplicación.

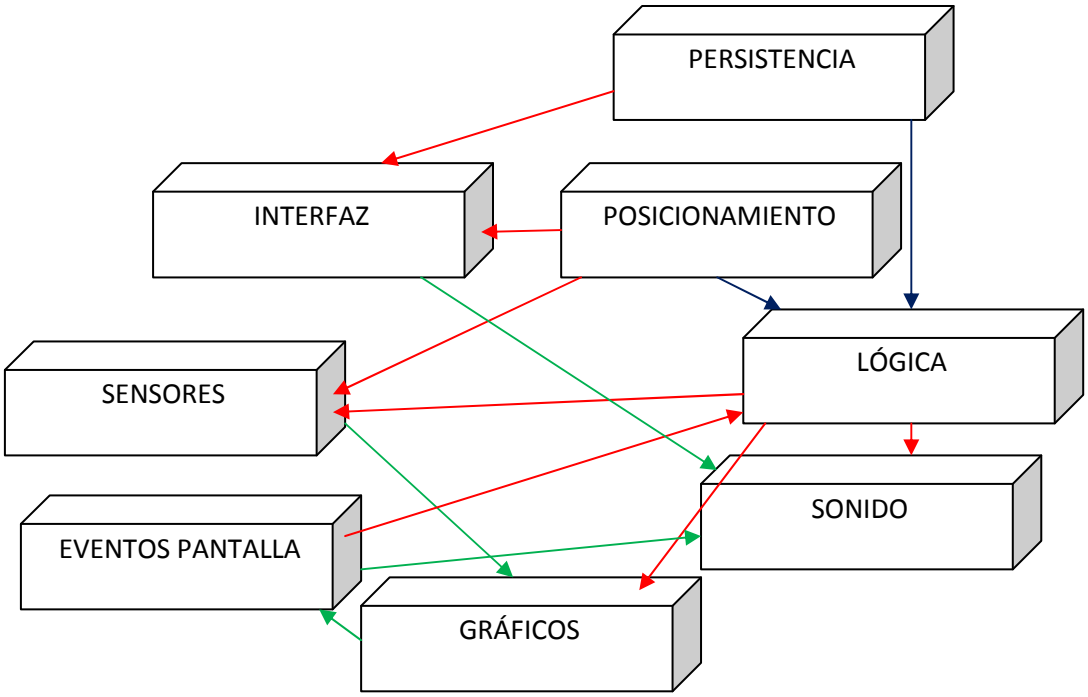


Imagen 46. Diagrama de interacción entre módulos, 2

- Interacción entre los módulos del sistema de realidad aumentada →
- Interacción entre los módulos incorporados por el videojuego *Invasión Androide* →
- Interacción entre módulos del sistema de realidad aumentada y los incorporados para la realización del videojuego →

A continuación se describirán brevemente los módulos que componen el sistema de realidad aumentada:

El módulo de *sensores* tendrá el principal cometido de pintar los objetos teniendo en cuenta que deberá pintarse sobre la imagen obtenida por la cámara del dispositivo del usuario atendiendo a la inclinación del terminal con respecto a la vertical y la horizontal de éste, además de la distancia con el usuario.

El módulo de *Gráficos* será el encargado de generar todos los objetos de visualización que requieran el openGL, así como de su correcto funcionamiento. Así pues, será el encargado de integrar los gráficos de los objetos y de los ítems.

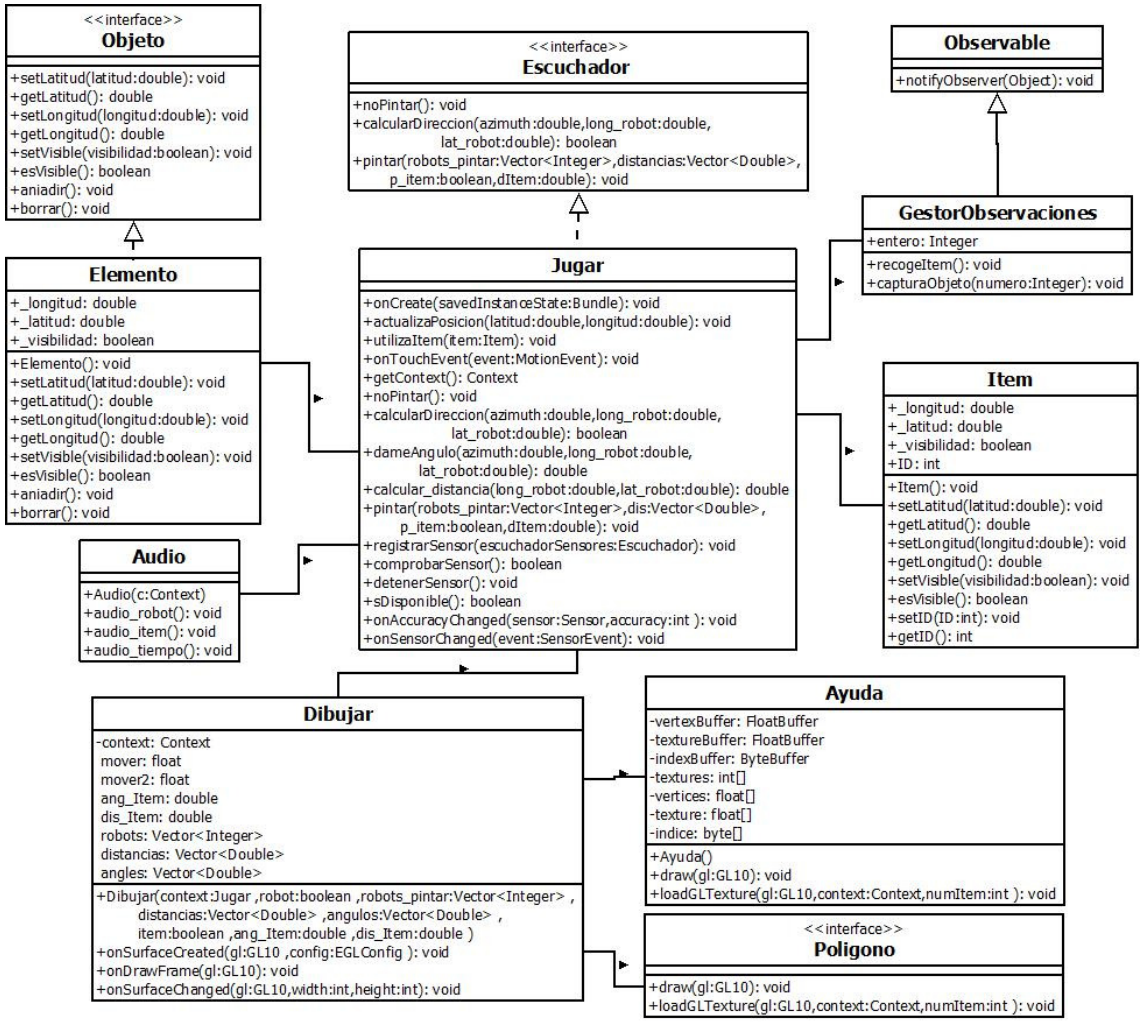
El módulo de *Sonido*, se encargará de proporcionar los diferentes efectos sonoros cuando así se requiera.

El módulo de *Eventos de pantalla*, controlará las pulsaciones sobre la pantalla táctil del dispositivo, comprobando si se pulsa sobre un objeto o un ítem que esté en condiciones de ser capturado.

3.3. Implementación

3.3.1. Contrato de interfaz

Para comenzar con el apartado de implementación en el que se explicará con detalle los pasos seguidos para la realización de este proyecto, es conveniente realizar un diagrama de clases de la parte del sistema que debe utilizarse para la realización de cualquier aplicación, es decir, las clases que participan en la interacción entre las aplicaciones y sus correspondientes métodos públicos.



Deben identificarse que clases corresponden a cada módulo citado en el apartado anterior, y así poder realizar una correcta implementación entre éstas y las clases correspondientes para el desarrollo de cualquier aplicación.

En el módulo de *sensores* estaría comprendida la clase *Jugar* así como el interfaz *Escuchador*, y las clases *Item* y *Elemento*. La clase *Jugar* es la clase principal del sistema, en la que se llevan a cabo las operaciones oportunas para determinar

mediante el uso de los sensores si deben o no pintarse los diferentes objetos y en qué posición y tamaño;

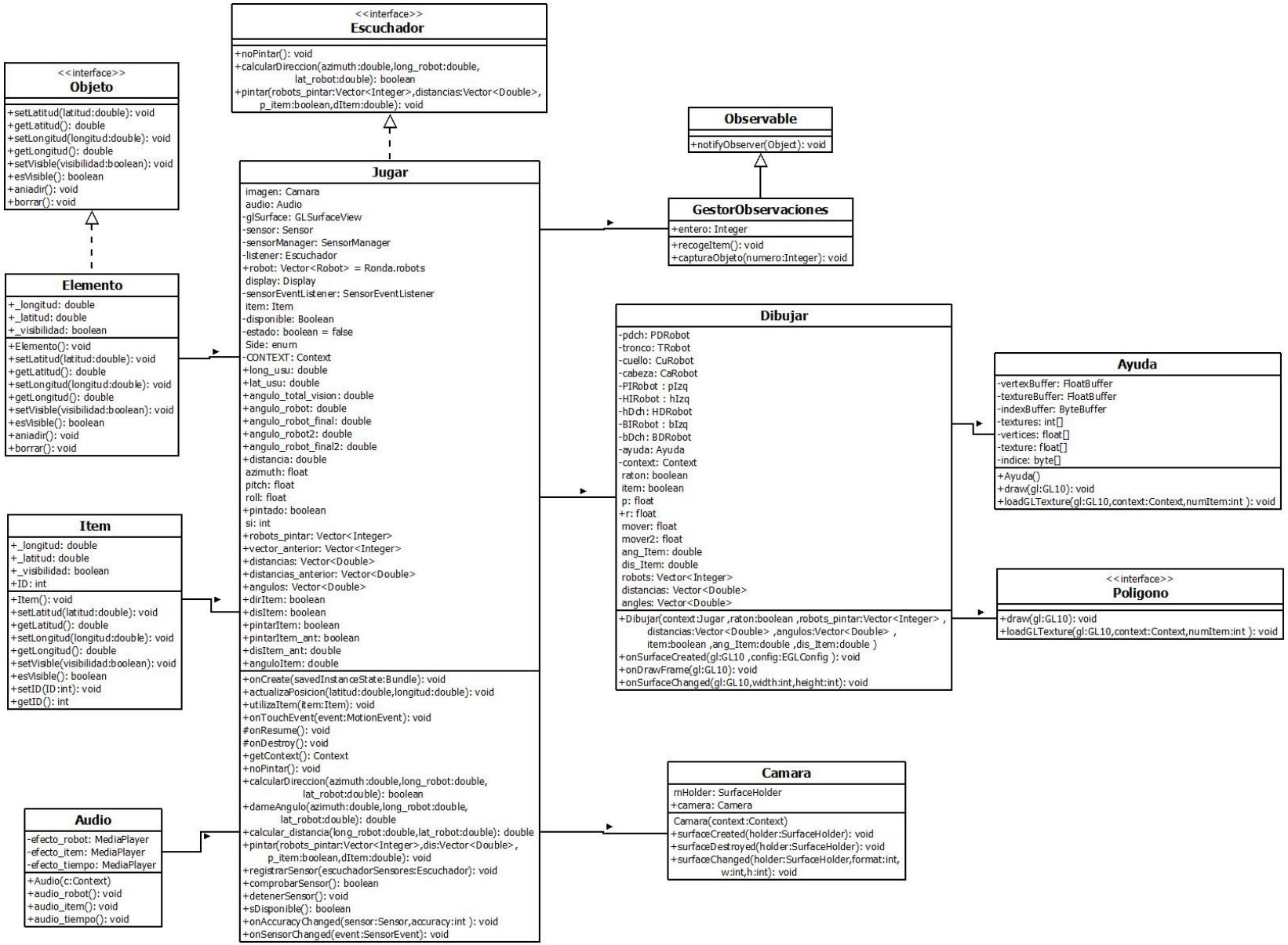
El módulo de gráficos se compone de las clases *Dibujar*, *Ayuda* y el interfaz *Polígono*. La clase *Dibujar* se encarga de mostrar los objetos OpenGL sobre la pantalla del dispositivo, y de aplicar las oportunas transformaciones a los objetos haciendo uso de la información obtenida por los sensores del dispositivo. La clase *Ayuda* define el polígono correspondiente al dibujo de los ítems, y el interfaz *Polígono* se corresponde con el interfaz que deben implementar todas las clases que quieran definir los objetos a pintar por el sistema. En nuestro caso concreto deberán implementar esta interfaz todas las clases correspondientes al dibujo de los robots.

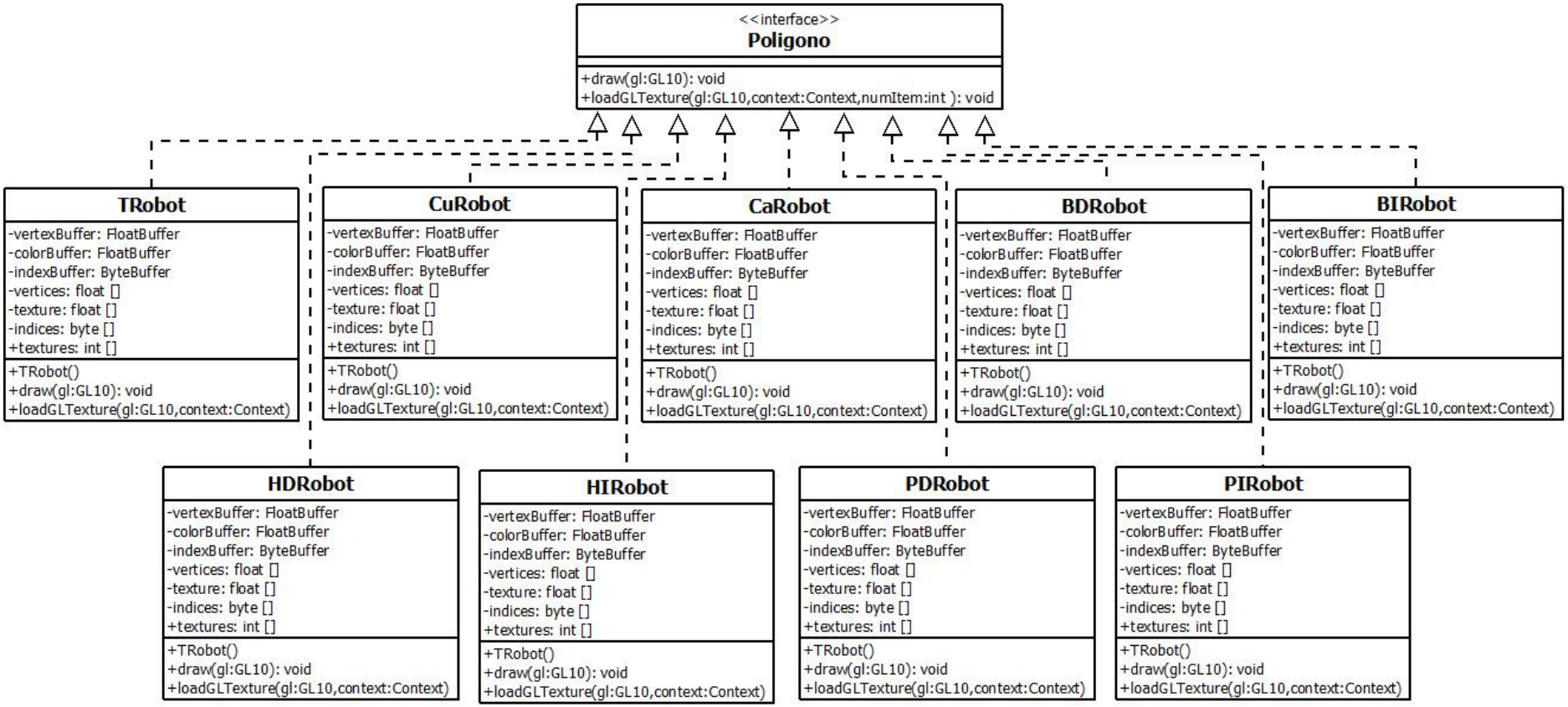
El módulo de *Audio* está comprendido por la clase *Audio* y en el módulo de *Eventos de Pantalla*, entran en juego la clases *Jugar* y *GestorObservaciones*.

Todas estas clases y las necesarias para completar el sistema de realidad aumentada son estudiadas con detalle en el apartado de Implementación.

3.3.2. Diagrama de clases

Finalmente, antes de introducirnos en los detalles de la implementación, se muestra un diagrama de clases del proyecto completo.


<<interface>>
Escuchador
+noPintar(): void
+calcularDireccion(azimuth:double,long_robot:double, lat_robot:double): boolean
+pintar(robots_pintar:Vector<Integer>,distancias:Vector<Double>, p_item:boolean,dItem:double): void
<<interface>>
Objeto
+setLatitud(latitud:double): void
+getLatitud(): double
+setLongitud(longitud:double): void
+getLongitud(): double
+setVisible(visibilidad:boolean): void
+esVisible(): boolean
+aniadir(): void
+borrar(): void
Observable
+notifyObserver(Object): void
Jugar
imagen: Camara
audio: Audio
-glSurface: GLSurfaceView
-sensor: Sensor
-sensorManager: SensorManager
-listener: Escuchador
+robot: Vector<Robot> = Ronda.robots
display: Display
-sensorEventListener: SensorEventListener
item: Item
-disponible: Boolean
-estado: boolean = false
Side: enum
-CONTEXT: Context
+long_usu: double
+lat_usu: double
+angulo_total_vision: double
+angulo_robot: double
+angulo_robot_final: double
+angulo_robot2: double
+angulo_robot_final2: double
+distancia: double
azimuth: float
pitch: float
roll: float
+pintado: boolean
si: int
+robots_pintar: Vector<Integer>
+vector_anterior: Vector<Integer>
+distancias: Vector<Double>
+distancias_anterior: Vector<Double>
+angulos: Vector<Double>
+dirItem: boolean
+disItem: boolean
+pintarItem: boolean
+pintarItem_ant: boolean
+disItem_ant: double
+anguloItem: double
+onCreate(savedInstanceState:Bundle): void
+actualizaPosicion(latitud:double,longitud:double): void
+utilizaItem(item:Item): void
+onTouchEvent(event:MotionEvent): void
#onResume(): void
#onDestroy(): void
+getContext(): Context
+noPintar(): void
+calcularDireccion(azimuth:double,long_robot:double, lat_robot:double): boolean
+dameAngulo(azimuth:double,long_robot:double, lat_robot:double): double
+calcular_distancia(long_robot:double,lat_robot:double): double
+pintar(robots_pintar:Vector<Integer>,dis:Vector<Double>, p_item:boolean,dItem:double): void
+registrarSensor(escuchadorSensores:Escuchador): void
+comprobarSensor(): boolean
+detenerSensor(): void
+sDisponible(): boolean
+onAccuracyChanged(sensor:Sensor,accuracy:int): void
+onSensorChanged(event:SensorEvent): void
GestorObservaciones
+entero: Integer
+recogeItem(): void
+capturaObjeto(numero:Integer): void
Elemento
+_longitud: double
+_latitud: double
+_visibilidad: boolean
+Elemento(): void
+setLatitud(latitud:double): void
+getLatitud(): double
+setLongitud(longitud:double): void
+getLongitud(): double
+setVisible(visibilidad:boolean): void
+esVisible(): boolean
+aniadir(): void
+borrar(): void
Dibujar
-pdch: PDRobot
-tronco: TRobot
-cuello: CuRobot
-cabeza: CaRobot
-PIRobot : pIzq
-HIRobot : hIzq
-hDch: HDRobot
-BIRobot : bIzq
-bDch: BDRobot
-ayuda: Ayuda
-context: Context
raton: boolean
item: boolean
p: float
+r: float
mover: float
mover2: float
ang_Item: double
dis_Item: double
robots: Vector<Integer>
distancias: Vector<Double>
angles: Vector<Double>
+Dibujar(context:Jugar ,raton:boolean ,robots_pintar:Vector<Integer> , distancias:Vector<Double> ,angulos:Vector<Double> , item:boolean ,ang_Item:double ,dis_Item:double)
+onSurfaceCreated(gl:GL10 ,config:EGLConfig): void
+onDrawFrame(gl:GL10): void
+onSurfaceChanged(gl:GL10,width:int,height:int): void
Ayuda
-vertexBuffer: FloatBuffer
-textureBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-textures: int[]
-vertices: float[]
-texture: float[]
-indice: byte[]
+Ayuda()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context,numItem:int): void
<<interface>>
Poligono
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context,numItem:int): void
Item
+_longitud: double
+_latitud: double
+_visibilidad: boolean
+ID: int
+Item(): void
+setLatitud(latitud:double): void
+getLatitud(): double
+setLongitud(longitud:double): void
+getLongitud(): double
+setVisible(visibilidad:boolean): void
+esVisible(): boolean
+setID(ID:int): void
+getID(): int
Audio
-efecto_robot: MediaPlayer
-efecto_item: MediaPlayer
-efecto_tiempo: MediaPlayer
+Audio(c:Context)
+audio_robot(): void
+audio_item(): void
+audio_tiempo(): void
Camara
mHolder: SurfaceHolder
+camera: Camara
Camara(context:Context)
+surfaceCreated(holder:SurfaceHolder): void
+surfaceDestroyed(holder:SurfaceHolder): void
+surfaceChanged(holder:SurfaceHolder,format:int, w:int,h:int): void


<<interface>>
Poligono
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context,numItem:int): void
TRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
CuRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
CaRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
BDRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
BIRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
HDRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
HIRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
PDRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)
PIRobot
-vertexBuffer: FloatBuffer
-colorBuffer: FloatBuffer
-indexBuffer: ByteBuffer
-vertices: float []
-texture: float []
-indices: byte []
+textures: int []
+TRobot()
+draw(gl:GL10): void
+loadGLTexture(gl:GL10,context:Context)

3.3.3. Detalles de implementación

En este apartado se explicarán con detalle los aspectos más importantes a la hora de la implementación del sistema. La estructura ha sido la misma que la que se siguió al programarlo.

Para dicha implementación podemos distinguir 4 áreas de trabajo principales que se han tenido que llevar a cabo:

- *Visualización de la imagen de la cámara*.
  Una de las partes más importantes en una aplicación de realidad aumentada, es acceder a la cámara del dispositivo para posteriormente poder mezclar esta imagen con los objetos virtuales.

- *Sensores*.
  Los sensores de movimiento con los que cuenta el teléfono nos proporcionarán una mayor sensación de realidad en el movimiento y posición de los objetos virtuales y ayudarán a determinar la dirección del usuario con respecto a dichos objetos.

- *OpenGL*.
  Mediante el estándar OpenGL diseñaremos tanto los objetos (en este caso los robots) como las ayudas disponibles en caso de querer hacer uso de éstas.

- *Sistema Observable/Observer.*

  Para poder dotar al sistema de total independencia a la hora de la captura de los objetos, vamos hacer uso del mecanismo Observable/Observer.

A parte de estas áreas han sido importantes otras con menor carga de trabajo como el audio, para añadir los efectos, y los eventos táctiles del teclado.

Veremos cómo se han implementado estos áreas y cómo interaccionan unas con otras para cumplir todas las funciones presentadas en los Casos de Uso.

Visualización de la Imagen de la Cámara

La imagen proporcionada por la cámara del dispositivo, es utilizada para mostrársela al usuario como fondo de pantalla y poder pintar encima los objetos OpenGL a capturar. Para visualizar la imagen obtenida, creamos la clase Camara, la cual proporcionará dicha imagen.

Clase Camara

La clase Camara extiende de SurfaceView e implementa SurfaceHolder.Callback. SurfaceView es un tipo de vista que contiene una superficie sobre la que dibujar, por lo que la utilizaremos para pintar la imagen de la cámara. La clase SurfaceHolder da acceso a la superficie donde pintar y la interfaz SurfaceHolder.Callback, contiene tres métodos que notifican cuando la superficie se crea, se destruye o cambia de tamaño.

En primer lugar creamos un objeto de la clase SurfaceHolder, y otro la de la clase Camera, la cual accede a la cámara del dispositivo. Como se muestra en el siguiente código, le indicamos al SurfaceHolder que notifique los cambios que se produzcan en la superficie a nuestro objeto, e indicamos el tipo de superficie a usar. Para la imagen de la cámara debemos utilizar el tipo *SURFACE_TYPE_PUSH_BUFFERS*.

```
Camara(Context context)
{
   super(context);

   mHolder = getHolder();
   mHolder.addCallback(this);
   mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
}
```

Código 01. Acceso a la cámara del dispositivo

Implementamos los 3 métodos correspondientes a la interfaz:

- public void surfaceCreated(SurfaceHolder holder): es llamado cuando la superficie se crea. Le indicamos que inicialice el sistema de la cámara y que muestre la imagen utilizando la superficie asociada al holder.

- public void surfaceDestroyed(SurfaceHolder holder): es llamado cuando la superficie se destruye, es decir, cuando cambiamos de actividad o iniciamos otra nueva. Paramos la visualización de la cámara y liberamos la cámara para

que pueda volver a ser utilizada si así se requiere. Si no la liberásemos y cambiásemos el foco principal de la aplicación a otra actividad, al volver a visualizar la imagen captada por la cámara provocaría excepciones de memoria y ésta no podría volver a ser usada.

- public void surfaceChanged(SurfaceHolder holder, int format, int w, int h): es llamado cuando la superficie cambia de tamaño, por ejemplo, cuando giramos el móvil de posición horizontal a vertical y viceversa. No utilizaremos los cambios de tamaño para girar la imagen ya que la imagen de la cámara en el videojuego no podrá ser girada en sentido horizontal. Aquí giramos la imagen 90º, ya que por defecto así se requiere, e inicializamos la visualización de la cámara.

```
public void surfaceCreated(SurfaceHolder holder)
{
  camera = Camera.open();
   try{
      camera.setPreviewDisplay(holder);
   }catch (IOException e) {
      camera.release();
      camera = null;
      e.printStackTrace();
   }
}

public void surfaceDestroyed(SurfaceHolder holder)
{
    camera.stopPreview();
    camera.release();
    camera = null;
}

public void surfaceChanged(SurfaceHolder holder, int format,
  int w, int h)
{
    Camera.Parameters parameters = camera.getParameters();
    parameters.set("orientation", "portrait");
    camera.setParameters(parameters);
    camera.startPreview();
}
```

Código 02. Implementación de los métodos de la interfaz SurfaceHolder.Callback

Cabe señalar que en el método `surfaceChanged` las líneas correspondientes al giro de la cámara, corresponden para una versión 2.1. Sin embargo, para versiones posteriores en las que se utiliza el sistema de GPS, como es el caso del videojuego *Invasión Androide* éstas deben ser sustituidas por `setDisplayOrientation(90)`.

Para poder utilizar la cámara, deben introducirse en el Manifiesto los oportunos permisos:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Sensores

Una de las principales novedades que introdujeron los Smartphone fue el uso de los sensores del teléfono, lo que supuso una revolución en el mundo de los videojuegos para dispositivos móviles. Utilizamos éstos para poder identificar hacia donde está apuntando la cámara del dispositivo y las rotaciones que se producen sobre éste, y poder así determinar si los objetos OpenGL se encuentran en el rango visible de la cámara y por tanto deben ser o no pintados.

Para manejar los sensores de un teléfono Android, hay que utilizar las clases Sensor y SensorManager. Actualmente el API de Android describe 11 tipos de sensores, pero la versión 2.1 (nivel 7) en la que está programado este proyecto, soporta únicamente 8 tipos:

- Sensor.TYPE_ACCELEROMETER: mide la aceleración de los 3 ejes en unidades de m/s^2.
- Sensor.TYPE_ORIENTATION: devuelve la orientación de los 3 ejes en grados. Analizaremos este sensor posteriormente.
- Sensor.TYPE_LIGHT: devuelve un valor que indica la cantidad de luz ambiental en unidades lux.
- Sensor.TYPE_MAGNETIC_FIELD: devuelve el campo magnético medido en los 3 ejes en unidades micro teslas (µT).
- Sensor.TYPE_PROXIMITY: devuelve un valor que indica la distancia en metros entre el dispositivo y un objeto destino.
- Sensor.TYPE_TEMPERATURE: devuelve un valor que indica la temperatura ambiente en grados Celsius (Cº).
- Sensor.TYPE_PRESSURE: devuelve un valor que indica la presión atmosférica.
- Sensor.TYPE_GYROSCOPE: mide la velocidad de rotación de los 3 ejes en unidades de radianes / segundo.

Para el caso que nos ocupa, necesitaremos el sensor de orientación proporcionado por el teléfono.

Los ejes son medidos de la misma manera en todos los dispositivos, en relación a la pantalla de éstos, como se indica en la siguiente figura:

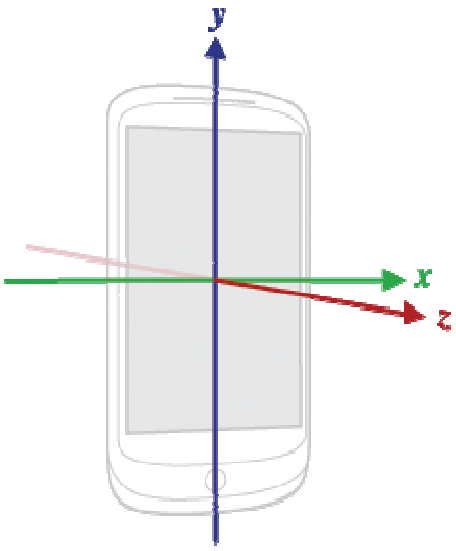


Imagen 47. Ejes dispositivo móvil

Mediante el sensor de orientación obtenemos 3 valores medidos en grados:

- Azimuth: mide el ángulo entre la dirección del norte magnético y el eje Z del dispositivo, siendo 0º el norte y 180º el sur. (Durante la elaboración del proyecto se tomará el norte magnético como norte geográfico puesto que la diferencia puede considerarse despreciable para el caso que nos ocupa.)

- Pitch: Mide la rotación del dispositivo sobre el eje X.

- Roll: Mide la rotación sobre el eje Y.

Se utilizará el valor azimuth para determinar la dirección de los robots con respecto a la orientación del dispositivo, y los valores de pitch y roll para pintar los objetos en función de la orientación de éste sobre los ejes X e Y.

Para obtener los valores de los sensores, una vez registrados éstos, hay que utilizar la interfaz SensorEventListener, e implementar sus métodos: *onAccuracyChanged(Sensor sensor, int accuracy)*, *onSensorChanged(SensorEvent event)*.

La tasa de actualización de los datos recogidos por los sensores, puede ser escogida entre cuatro valores predeterminados:

- SensorManager.SENSOR_DELAY_FASTEST: indica la mayor tasa posible de actualización.

- SensorManager.SENSOR_DELAY_GAME: indica una actualización adecuada para el control de videojuegos.

- SensorManager.SENSOR_DELAY_NORMAL: indica la tasa de actualización por defecto.

- SensorManager.SENSOR_DELAY_UI: indica una tasa adecuada para una interfaz de usuario.

Dado que el objetivo de este proyecto es la creación de un sistema que será principalmente utilizado para la creación de videojuegos, se utilizará la tasa de actualización especificada para ello.

Para poder hacer uso del sensor de orientación, lo primero que debemos hacer es ver si éste está disponible y registrarlo indicándole la tasa de actualización que deseamos.

```
  public static boolean sDisponible() {
     if (disponible == null) {
        if (Jugar.getContext() != null) {
           sensorManager = (SensorManager) Jugar.getContext()
                   .getSystemService(Context.SENSOR_SERVICE);
           List<Sensor> sensors = sensorManager.getSensorList(
                    Sensor.TYPE_ORIENTATION);
           disponible = new Boolean(sensors.size() > 0);
           } else {
             disponible = Boolean.FALSE;
         }
      }
      return disponible;
  }
```

Código 03. Obtención de la disponibilidad del sensor de orientación.

```
  public static void registrarSensor(Escuchador
escuchadorSensores)
 {
    sensorManager = (SensorManager) getContext()
           .getSystemService(Context.SENSOR_SERVICE);
    List<Sensor> sensors = sensorManager.getSensorList(
           Sensor.TYPE_ORIENTATION);
    if (sensors.size() > 0) {
            sensor = sensors.get(0);
            estado = sensorManager.registerListener
                      (sensorEventListener, sensor,
                           SensorManager.SENSOR_DELAY_GAME);
```

```
                listener = escuchadorSensores;
    }
}
```

Código 04. Registro del sensor de orientación.

Coordenadas Geográficas

Un aspecto importante para la realización del proyecto es tratar de calcular mediante las coordenadas geográficas del usuario y de los robots, en qué dirección se encuentran éstos y comprobar si el teléfono está apuntando en esa misma dirección para poder pintarlos. Es aquí donde entra el primer valor del sensor del que se ha hablado con anterioridad: azimuth.

Como ya se había comentado, el azimuth corresponde con el valor en grados a los que se encuentra apuntando el dispositivo con respecto al norte. Haciendo uso de las fórmulas de rumbo loxodrómico entre dos puntos dadas dos coordenadas geográficas (en nuestro caso la del usuario y la del robot), podemos calcular en qué dirección (grados con respecto al norte) se encuentra la segunda de ellas.

$$\tan Rlox = \frac{\Delta\lambda}{\Delta\varphi} \times \cos \varphi_{media}$$

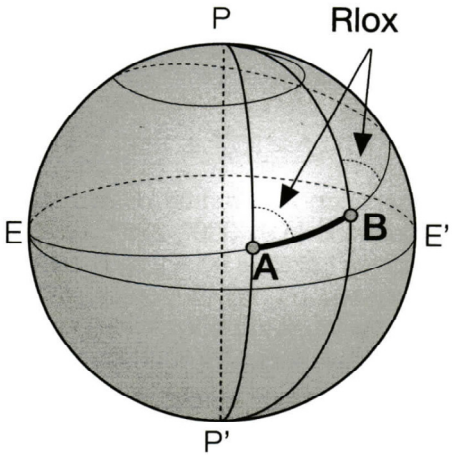


Imagen 48. Rumbo loxodrómico

Donde:

$\Delta\lambda$ es el incremento de la longitud geográfica entre el usuario y los objetos.

$\Delta\phi$ es el incremento de la latitud geográfica entre el usuario y los objetos.

ϕ_{media} es la latitud media entre el usuario y los objetos.

Dependiendo del cuadrante en el que se encuentre el objeto a capturar, a la medida del ángulo devuelto a veces habrá que sumarle 180 o 360 grados.

Una vez obtenidos los grados a los que se encuentra el objeto con respecto al norte, junto con el valor del azimuth y tomando que la amplitud del ángulo de visor de la cámara del móvil es de 60º, se puede determinar en qué momentos el objeto debe ser visualizado y en cuáles no.

```
public boolean calcularDireccion(double azimuth, double
long_objeto, double lat_objeto){
      double longitud=long_objeto – long_usu;
      longitud = Math.toRadians(longitud);
      double latitud =lat_objeto – lat_usu;
      latitud = Math.toRadians(latitud);
      double lat_med=(lat_usu + lat_ objeto)/2;
      lat_med = Math.toRadians(lat_med);
      angulo_objeto = Math.atan ((longitud/latitud)*
      Math.cos(lat_med));
      angulo_objeto = Math.toDegrees (angulo_objeto);
      double visor = angulo_total_vision/2;
      boolean seVe =false;
      if (long_objeto > long_usu){
            if (lat_objeto > lat_usu){
                  angulo_objeto_final  = angulo_objeto ;
            }else{
                  angulo_objeto_final  = angulo_objeto + 180;
            }
      }else{
            if (lat_objeto > lat_usu){
                  angulo_objeto_final  = angulo_objeto + 360;
            }else{
                  angulo_objeto_final  = angulo_objeto + 180;
            }
      }if(azimuth>=0&&azimuth<=30){
            if (angulo_objeto_final<azimuth +visor ||
                        angulo_objeto_final > (azimuth-visor)+360)
                        seVe=true;
            }
      if(azimuth>=330&&azimuth<=360){
            if (angulo_objeto_final>azimuth – visor ||
                              angulo_objeto_final <
(azimuth+visor)-360)
                        seVe=true;
            }
      if(azimuth>30&&azimuth<330){
            if (angulo_objeto_final >azimuth – visor &&
                        angulo_objeto_final < azimuth + visor)
                        seVe=true;
            }
      if(seVe==true)
                  return true;
      else
                  return false;

}
```

Código 05. Método que calcula si un objeto se encuentra o no en la dirección apuntada por el dispositivo móvil.

De la misma manera que se ha calculado el ángulo, puede calcularse la distancia a la que se encuentra los objetos mediante la fórmula:

$$Dlox = \frac{\Delta\lambda}{\text{sen } Rlox} \text{ x cos } \varphi_{media}$$

Donde:

$\Delta\lambda$ es el incremento de la longitud geográfica entre el usuario y los objetos.
R_{lox} es el rumbo loxodrómico calculado anteriormente.
ϕ_{media} es la latitud media entre el usuario y los objetos.

La distancia se tendrá en cuenta para:

1. Sólo se visualizará un objeto si éste se encuentra a menos de 20 metros del usuario.

2. Sólo se podrá capturar un objeto si éste se encuentra a menos de 7 metros del usuario.

3. Según la distancia a la que se encuentren, se pintarán de un determinado tamaño (cuanto más lejos, más pequeños).

4. Según la distancia a la que se encuentren se pintarán a una determinada altura de la pantalla (cuanto más lejos, más arriba).

```
public static double calcular_distancia(double long_raton,
double lat_raton)
{
    double longitud=long_raton - long_usu;
    longitud = Math.toRadians(longitud);
    double lat_med=(lat_usu + lat_raton)/2;
    lat_med = Math.toRadians(lat_med);
    double angulo =  Math.toRadians(angulo_objeto_final);
    distancia = longitud / Math.sin (angulo) *
                    Math.cos(lat_med);
    distancia =  Math.toDegrees (distancia)* 60 * 1852;
    return distancia;

}
```

Código 06. Método que calcula la distancia entre el usuario y los objetos

La distancia obtenida directamente por las fórmulas se corresponde al valor en grados de arco de círculo máximo, por lo que para obtener el resultado en metros habrá que multiplicar por 60 (para pasar de grados a minutos – un minuto de arco de

círculo máximo corresponde a una milla náutica-) y por 1852 (metros de una milla náutica).

La posición a la que se encuentra el usuario debe actualizarse continuamente para poder calcular a qué distancia y en que ángulo se encuentra éste respecto a los robots. Para ello debe llamarse al método *actualizaPosicion*, introduciendo como parámetros la longitud y latitud del usuario.

```
public static void actualizaPosicion(double longitud,
double latitud)
{
      long_usu=longitud;
      lat_usu=latitud;
}
```

Código07. Actualización de la posición del usuario

Interface Objeto

De la misma manera, se necesita conocer la posición de los objetos a pintar, en nuestro caso concreto de los robots. Para ello la clase encargada de crear dichos objetos deberá implementar la interfaz Objeto y sus correspondientes métodos, los cuales indican la longitud y latitud de éstos, si es visible desde la posición del usuario, y dos métodos que permitan añadirlo y borrarlo al array correspondiente.

```
public interface Objeto
{
    public void setLatitud(double latitud);
    public double getLatitud();

    public void setLongitud(double longitud);
    public double getLongitud();

    public void setVisible(boolean visibilidad);
    public boolean esVisible();

    public void aniadir ();
    public void borrar();
}
```

Código 08. Interfaz Objeto

Por tanto, clase creada para la creación de los objetos que se deseen visualizar, deberá será del tipo:

```
public class Elemento implements Objeto
{

    private double _longitud;
    private double _latitud;
    private boolean _visible;

    public Elemento(){

          _visible = false;

    }

    public void setLatitud(double latitud){
          _latitud = latitud;
    }

    public double getLatitud() {
          return _latitud;
    }

    public void setLongitud(double longitud) {
          _longitud = longitud;
    }

    public double getLongitud() {
          return _longitud;
    }

    public boolean esVisible(){
          return _visible;
    }

    public void setVisible(boolean visibilidad){
          _visible = visibilidad;
    }

    public void aniadir() {
          Jugar.objetos.add(this);
    }

    public void borrar() {
          Jugar.objetos.remove(this);
    }
```

Código 09. Clase Elemento

Clase Item

Si se desea utilizar la opción proporcionada por el sistema de la inclusión de ítem que permitan dar una mayor sensación de videojuego, se debe, como en el caso de los robots, indicar las coordenadas en las que se encuentran, y además, qué tipo de ítem es (caracterizado por un ID) y si se está visualizando o no.

```
public class Item
{
	private double _longitud;
	private double _latitud;
	private boolean _visible;
	private int _ID;

	public Item(){
		_ID = (int) (Math.random()*5);
	}

	public void setLongitud(double longitud) {
		_longitud = longitud;
	}

	public double getLongitud() {
		return _longitud;
	}

	public void setVisible(boolean visible) {
		_visible = visible;
	}

	public boolean isVisible() {
		return _visible;
	}

	public void setLatitud(double latitud) {
		_latitud = latitud;
	}

	public double getLatitud() {
		return _latitud;
	}

	public void setID(int ID) {
		_ID = ID;
	}

	public int getID() {
		return _ID;
	}

}
```

Código 10. Clase Item

Para indicar al sistema que se desea utilizar ítems, hay que llamar al método *utilizaItem* y pasar como parámetro el objeto de la clase Ítem que se desea utilizar.

```
public static void utilizaItem(Item it)
{
      item = new Item();
      item = it;
}
```

Código 11. Indicar el uso de ítems.

Sensores y coordenadas geográficas

Para determinar cuándo ha de repintarse la pantalla, es decir, cuándo deben visualizarse los objetos (robots) o las ayudas, utilizamos los apartados de sensores y coordenadas explicados previamente.

Para pintar elementos en la pantalla debemos comprobar:

- para cada robot:
  1. Si está en la dirección a la que apunta el móvil.
  2. Si es visible desde la posición en la que se encuentra el usuario.
  3. Si está a una distancia menor o igual a 20 metros.

- Para los ítems:
  1. Si existe algún ítem en ese momento disponible
  2. Si está en la dirección a la que apunta el móvil.
  3. Si es visible desde la posición en la que se encuentra el usuario.
  4. Si está a una distancia menor o igual a 20 metros.

Todas estas comprobaciones las haremos desde el método *public void onSensorChanged(SensorEvent event)*, es decir, cada vez que se recogen nuevamente los valores de orientación del dispositivo. Sin embargo, para no repintar la pantalla innecesariamente, antes de llamar al método pintar(), tendremos en cuenta que:

- Los robots a pintar sean distintos que los que están visualizados en ese momento en pantalla,

- o que las distancias de los robots que están pintados hayan sido modificadas,

- o que la visualización de un ítem haya cambiado.

Para ello almacenamos en vectores los robots a pintar y sus respectivas distancias, y en una variable *boolean* si hay un ítem pintado o no, y comprobamos cada vez que el método *onSensorChanged* es llamado que éstas variables no son las mismas que en la llamada anterior.

```
   public void onSensorChanged(SensorEvent event)
   {
      azimuth = event.values[0];      // azimuth
      pitch = event.values[1];     // pitch
      roll = event.values[2];        // roll

      objetos_pintar.clear();
      distancias.clear();
      for (int x=0;x<objetos.size();x++){
         if(listener.calcularDireccion(azimuth,
                (objetos.elementAt(x).getLongitud())/1E6,

      (objetos.elementAt(x).getLatitud())/1E6)==true)
         {
            double d=0;
            double dis = calcular_distancia

      ((objetos.elementAt(x).getLongitud())/1E6,
           (objetos.elementAt(x).getLatitud())/1E6);
            if(dis<=20 && objetos.elementAt(x).esVisible()){
                objetos_pintar.add(x);
                if (dis <20 && dis>=15){
                     d=19;
                }if (dis <15 && dis>=10){
                     d=12;
                }if (dis < 10 && dis>=7){
                     d=8;
                }if (dis<7 && dis>4){
                     d=5;
                }if (dis<=4 && dis>=2){
                     d=3;
                }if (dis<2 && dis>=0){
                     d=1;
                }
                distancias.add(d);
            }
         }
       }
       si=objetos_pintar.size();
       if(item!=null && item.isVisible()==true){
          dirItem=listener.calcularDireccion(azimuth,
                (item.getLongitud())/1E6,
(item.getLatitud())/1E6);
          if (dirItem ==true){
            disItem= calcular_distancia
                 ((item.getLongitud())/1E6,
(item.getLatitud())/1E6);
            if(disItem<=20){
                pintarItem=true;
            }
            else {
                pintarItem=false;
            }
          }else{
                pintarItem=false;
          }
```

```
       }else{
            pintarItem=false;
        }

       if(si>0 || pintarItem==true){
            currentSide = Side.PINTAR;
        }
       else{
            currentSide = Side.NOPINTAR;
       }
       if (currentSide != null){
            switch (currentSide) {
                        case PINTAR :
      if(!vector_anterior.equals(objetos_pintar)||!distancias_ant
erior.equals(distancias)||pintarItem_ant!=pintarItem||disItem_an
t!=disItem){

      listener.pintar(objetos_pintar,distancias, pintarItem,
disItem);
          }
                        break;
                        case NOPINTAR:
if(!currentSide.equals(oldSide))
      listener.noPintar();

                        break;
     }
          oldSide = currentSide;
          pintarItem_ant=pintarItem;
          disItem_ant=disItem;
          vector_anterior.clear();
          distancias_anterior.clear();
          if(si>0){
              for (int a=0;a<objetos_pintar.size();a++){
                 vector_anterior.add(a, objetos_pintar.get(a));
              }
              for (int b=0;b<distancias.size();b++){
                 distancias_anterior.add(b, distancias.get(b));
              }
          }
   }
 }
```

Código 12. Comprobación de los objetos a pintar según la orientación del dispositivo

OpenGL

Para dibujar tanto los ítems como los robots se ha utilizado el estándar OpenGL. Ha de apuntarse que en un principio, cuando se pensó la idea del videojuego, se iban a capturar ratones en vez de robots. Sin embargo cuando se empezó a trabajar con OpenGL y se conoció la complejidad de pintar un ratón en 3D y que diese sensación de realidad a la hora de girar el dispositivo, hubo que buscar otras alternativas para darle un mayor realismo al videojuego. Finalmente se optó por utilizar un robot, creado a partir de polígonos, como se verá a lo largo de este apartado.

Interfaz Polígono

Todos los polígonos que conforman la imagen del robot implementan la interfaz Polígono, la cual consta de los métodos:

- draw (GL gl10)
- loadGlTexture (GL10 gl, Context context)

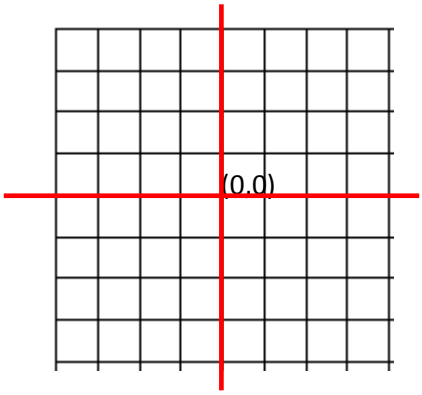
```
public interface Poligono
{
      public void draw(GL10 gl);
      public void loadGLTexture(GL10 gl, Context context);
}
```

Código 13. Interfaz Polígono

Para crear cualquier elemento que se vaya a integrar en el sistema, éste debe implementar la interfaz Polígono y todos sus métodos.

Para construir polígonos se definen una serie de vértices que, unidos unos con otros, denotarán nuestro polígono final. Para construir los polígonos que conformarán el robot definiremos 24 vértices por cada polígono, 4 por cada lado del rectángulo, cada vértice con su valor de altura, anchura y profundidad.

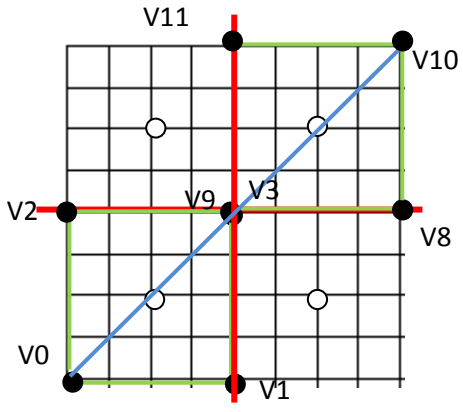
Para entenderlo visualmente, OpenGL divide la superficie donde dibujar en una cuadrícula que toma el valor 0,0 en el centro.



Definimos los 24 vértices y los unimos de manera que queden definidas las 6 caras del polígono. Por ejemplo, para la caras delantera y trasera de la cabeza del robot:

- -0.6f, 2.0f, 1.0f, (V0)
- 0.6f, 2.0f, 1.0f, (V1)
- -0.6f, 3.2.f, 1.0f, (V2)
- 0.6f, 3.2f, 1.0f, (V3)

- 0.6f, 2.0f, -1.0f, (V8)
- -0.6f, 2.0f, -1.0f, (V9)
- 0.6f, 3.2.f, -1.0f, (V10)
- -0.6f, 3.2f, -1.0f, (V11)



Todas las partes del robot quedan definidas mediante las clases TRobot, CaRobot, CuRobot, HDRbotot, HIRobot, BDRobot, BIRobot, PDRobot y PDRobot, siendo respectivamente el cuerpo, la cabeza, el cuello, los hombros, los brazos y las piernas del robot.

Texturas

OpenGL permite añadir texturas a los objetos para poder añadirles imágenes. En el caso de los polígonos del robot se les ha agregado una imagen .bmp. Implementamos el método *loadGLTexture(GL10 gl, Context context)*, con el que añadiremos a los polígonos generados anteriormente la imagen que corresponda a cada parte del cuerpo del robot.

```
public void loadGLTexture(GL10 gl, Context context) {

      Bitmap bitmap =

      BitmapFactory.decodeResource(context.getResources(),
            R.drawable.cuello);

      gl.glGenTextures(1, textures, 0);
      gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);

      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);

      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);

      GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);

      bitmap.recycle();
}
```

Código 14.Textura del cuello del robot

Clase Ayuda

La clase Ayuda define el polígono cubo que utilizaremos para representar las ayudas que el usuario podrá ir cogiendo a lo largo de la partida.

Para crear los ítems procedemos de la misma manera que para las partes del robot, con una salvedad, ya que a la hora de cargar la textura de los ítems se debe indicar el identificador de éste para añadirle una u otra textura, mediante el parámetro *numItem*, del método *loadGLTexture(GL10 gl, Context context, int numItem)*,

El sistema permite añadir 3 texturas diferentes a los ítems.

```
public void loadGLTexture(GL10 gl, Context context, int numItem)
{
      InputStream is;
      if (numItem==0){
            is =

      context.getResources().openRawResource(R.drawable.item_a);
      }
      else if (numItem==1){
            is =
      context.getResources().openRawResource(R.drawable.item_b);
      }
      else{
            is =
      context.getResources().openRawResource(R.drawable.item_c);
      }
      Bitmap bitmap = null;
      try {
            bitmap = BitmapFactory.decodeStream(is);

      } finally {
            try {
                  is.close();
                  is = null;
            } catch (IOException e) {
            }
      }

      gl.glGenTextures(1, textures, 0);
      gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                  GL10.GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);
```

```
        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
        bitmap.recycle();
}
```

Código 15. Método que carga la textura correspondiente a un ítem.

Clase Dibujar

La clase Dibujar implementa *android.opengl.GLSurfaceView.Renderer*, interfaz genérica para la representación de objetos. Dicha interfaz consta de tres métodos:

- onSurfaceCreated(GL10 gl, EGLConfig config): se llama cada vez que la superficie de dibujo se crea.

- onSurfaceChanged(GL10 gl, int width, int height): se llama cuando cambia el tamaño de la superficie de dibujo.

- onDrawFrame(GL10 gl): se llama desde el hilo creado por la clase GLSurfaceView que se explicará posteriormente. Aquí es donde se pintan los objetos OpenGL y donde se indican las transformaciones.

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
      if(item)
      {
            ayuda.loadGLTexture(gl, this.context,
                  Jugar.item.getID());
      }
      if(objeto)
      {
            pdch.loadGLTexture(gl, this.context);
            tronco.loadGLTexture(gl, this.context);
            cuello.loadGLTexture(gl, this.context);
            cabeza.loadGLTexture(gl, this.context);
            pIzq.loadGLTexture(gl, this.context);
            bIzq.loadGLTexture(gl, this.context);
            hIzq.loadGLTexture(gl, this.context);
            bDch.loadGLTexture(gl, this.context);
            hDch.loadGLTexture(gl, this.context);
      }
      gl.glEnable(GL10.GL_TEXTURE_2D);
      gl.glShadeModel(GL10.GL_SMOOTH);
      gl.glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
      gl.glClearDepthf(1.0f);
      gl.glEnable(GL10.GL_DEPTH_TEST);
      gl.glDepthFunc(GL10.GL_LEQUAL);
      gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
                  GL10.GL_NICEST);
```

Código 16. Método onSurfaceCreated

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
      if(height == 0) {
            height = 1;
      }
      gl.glViewport(0, 0, width, height);
      gl.glMatrixMode(GL10.GL_PROJECTION);
      gl.glLoadIdentity();

      GLU.gluPerspective(gl, 45.0f, (float)width /
            (float)height, 0.1f, 100.0f);

      gl.glMatrixMode(GL10.GL_MODELVIEW);
      gl.glLoadIdentity();
}
```

Código 17. Método onSurfaceChanged

Transformaciones

Podemos utilizar tres tipos de transformaciones a los objetos diseñados con OpenGL. Veamos cómo podemos aplicarlas para cumplir los requisitos del proyecto y darle una mayor sensación de realidad a los objetos 3D.

- *Traslación*

public abstract void glTranslatef (float x, float y, float z)

Mediante la traslación movemos todos los vértices de un polígono tantas unidades en cada eje como indiquemos en la función.

En nuestro caso queremos que nuestros objetos (los robots y los ítems) se desplacen en el eje X según la posición del teléfono, es decir, según el ángulo que forme el teléfono en cada momento.

Para saber en qué lugar del eje X a lo largo de la pantalla debemos pintar, aplicamos la siguiente función:

```
if(angles.elementAt(num)>330&&angles.elementAt(num)<360&&Jugar.a
zimuth<30){
      mover2 = (float) ((angles.elementAt(num)-  (Jugar.azimuth+
                  360))/30);
}if(angles.elementAt(num)>0&&angles.elementAt(num)<30&&Jugar.azi
muth>330){
      mover2 = (float) ((angles.elementAt(num)-(Jugar.azimuth-
                  360))/30);
}else
      mover2 =(float) ((angles.elementAt(num)/Jugar.azimuth)/30);
```

Código 18. Pintar sobre el eje X

Donde:

- *angles* es un vector donde están almacenados los ángulos con respecto al norte que forma cada robot a pintar.
- *Jugar.azimuth* es el ángulo azimuth del teléfono explicado en capítulos anteriores.
- *Mover2* lugar de la pantalla dónde hay que pintar al robot.

Imagen 49. Traslación en el eje X.

La traslación en el eje Y será utilizada para dar sensación de lejanía o cercanía de los objetos, es decir, según la distancia en la que se encuentran, serán pintados en una altura diferente de la pantalla. Cuanto más lejos se encuentren, más arriba de la pantalla serán visualizados, y cuanto más cerca, más abajo.

Imagen 50. Traslación en el eje Y.

- Escalado

public abstract void glScalef (float x, float y, float z)

Mediante el escalado aumentamos o disminuimos el tamaño de nuestros polígonos, es decir, multiplicamos todos sus vértices por el mismo valor. Podemos aplicar el escalado en todos los ejes, y debemos aplicar el mismo en todos ellos para no deformar el polígono construido inicialmente. Al igual que con la traslación, escalaremos nuestros objetos según la distancia a la que se encuentren haciendo éstos cada vez mas grandes según se vayan acercando al dispositivo.

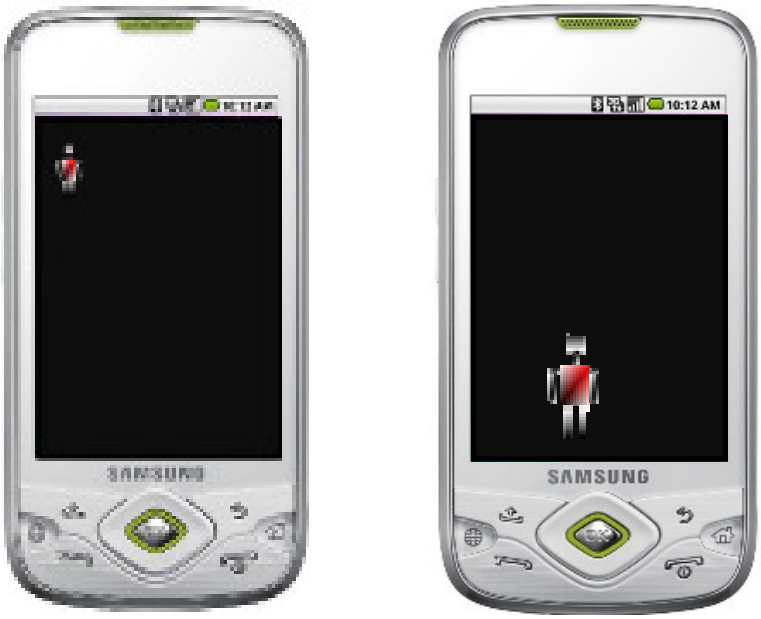


Imagen 51. Escalado

- *Rotación*

public abstract void glRotatef(float angle, float x, float y, float z)

Mediante la rotación podemos hacer girar nuestros objetos en el eje que queramos tanto grados como indiquemos. Aquí es donde entran en escena los valores de pitch y roll captados por el sensor de orientación que no habíamos utilizado hasta el momento. Si recordamos, pitch nos indica la rotación del dispositivo en el eje X y roll la rotación en el eje Y.

De esta manera, nuestro método onDrawFrame(GL10 gl) quedaría:

```
public void onDrawFrame(GL10 gl){

      gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                GL10.GL_DEPTH_BUFFER_BIT);
      gl.glLoadIdentity();
      if (raton){
            for(int num=0;num<robots.size();num++){

      if(angles.elementAt(num)>330&&angles.elementAt(num)<360&&
                Jugar.azimuth<30){
                mover2 = (float) ((angles.elementAt(num)-
                          (Jugar.azimuth+360))/30);
      }

      if(angles.elementAt(num)>0&&angles.elementAt(num)<30&&
                Jugar.azimuth>330){
                mover2 = (float) ((angles.elementAt(num)-
                          (Jugar.azimuth-360))/30);
      }
      else
            mover2 =(float) ((angles.elementAt(num)
                      Jugar.azimuth)/ 30);

      if (distancias.get(num) <20 && distancias.get(num)>=15){
            gl.glTranslatef(mover2 ,0.8f , -3.0f);
            gl.glScalef(0.01f, 0.01f, 0.01f);
      }
      if (distancias.get(num) <15 && distancias.get(num)>=10){
            gl.glTranslatef(mover2 ,0.6f , -3.0f);
            gl.glScalef(0.02f, 0.02f, 0.02f);

      }
      if (distancias.get(num) < 10 && distancias.get(num)>=7){
            gl.glTranslatef(mover2 ,0.3f , -3.0f);
            gl.glScalef(0.04f, 0.04f, 0.04f);
      }
      if (distancias.get(num) <7 && distancias.get(num)>4){
            gl.glTranslatef(mover2 ,-0.2f , -3.0f);
            gl.glScalef(0.08f, 0.08f, 0.08f);
      }
      if (distancias.get(num)<=4 && distancias.get(num)>=2){
            gl.glTranslatef(mover2 ,-0.4f , -3.0f);
            gl.glScalef(0.09f, 0.09f, 0.09f);
      }
      if (distancias.get(num) <2 && distancias.get(num)>=0){
            gl.glTranslatef(mover2 ,-0.6f , -3.0f);
            gl.glScalef(0.1f, 0.1f, 0.1f);
      }

            p= Jugar.pitch;
            r=Jugar.roll;

            gl.glRotatef(p-270, 1.0f, 0.0f, 0.0f);
            gl.glRotatef(r, 0.0f, 1.0f, 0.0f);
```

```
            tronco.draw(gl);
            pdch.draw(gl);
            cuello.draw(gl);
            cabeza.draw(gl);
            pIzq.draw(gl);
            hDch.draw(gl);
            hIzq.draw(gl);
            bDch.draw(gl);
            bIzq.draw(gl);

            gl.glLoadIdentity();

            }
      }

  if (item){

      //…
 }
}
```

Código 19. Método onDrawFrame

Observable / Observer

Para poder dotar al sistema de total independencia a la hora de la captura de los objetos vamos hacer uso del mecanismo Observable/Observer, donde Observable es un objeto cuyo comportamiento queremos observar, y Observer, un objeto que está pendiente del comportamiento de algún Observable. Por lo tanto, una clase que extiende de Observable se encarga de notificar a todos los objetos dependientes de ella (observadores) los cambios de estado. En nuestro caso particular, vamos a implementar la clase GestorObservaciones, que extienda de Observable, y se encargue de notificar a todos los observadores registrados cuando se ha producido la captura de un objeto o de un ítem.

```
public class GestorObservaciones extends Observable
{
      Integer entero;

      public void recogeItem(){
            entero =-1;
            setChanged();
            notifyObservers(entero);
      }
      public void capturaObjeto(Integer numero){
            setChanged();
            notifyObservers(numero);
      }
}
```

Código20. Implementación de un Observable

De esta manera todos los observadores registrados serán notificados recibiendo un -1 cuando se haya capturado un ítem, y recibiendo el número correspondiente a la posición de un robot dentro del array cuando alguno de éstos haya sido capturado.

Para poder hacer uso del Observable cualquier aplicación que utilice el sistema deberá:

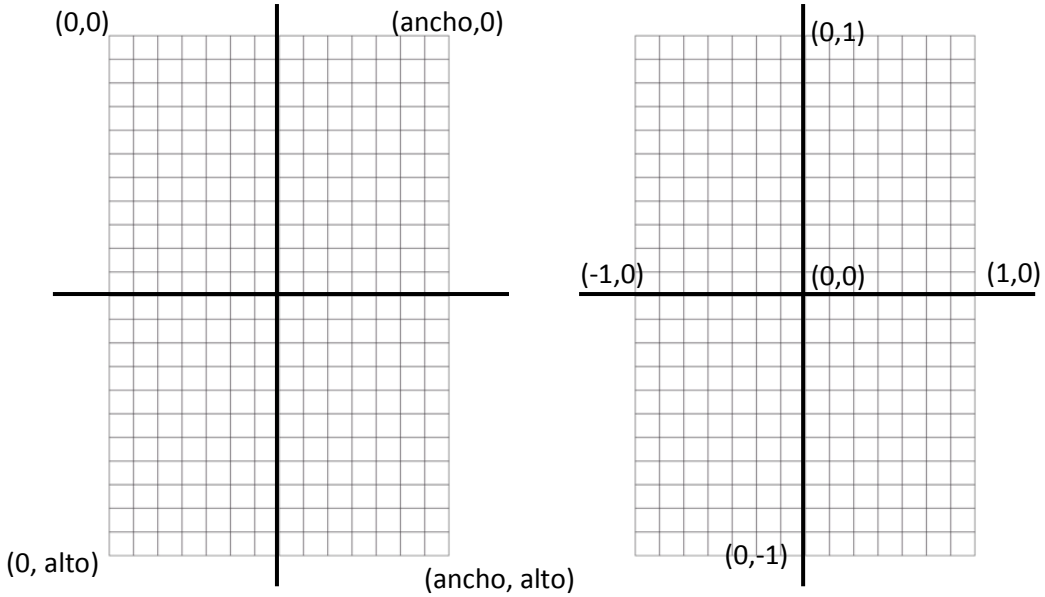
1. Crear una clase que implemente la interfaz *Observer* y su correspondiente método *public void update(Observable arg0, Object arg1)*, donde arg0 corresponde con el Observable del que proviene la notificación y arg1 el objeto recibido, en nuestro caso el entero que indica la captura de un ítem (-1) o de la posición dentro del array del robot capturado, para así poder llevar a cabo las acciones correspondientes a cada captura.

2. Registar el Observer creado añadiendo esté a nuestro Observable mediante el método *Observable.addObserver(Observer).*

Eventos táctiles

Para controlar cuando el usuario ha pulsado sobre un robot o una ayuda que podían ser capturados vamos a controlar los eventos táctiles que se producen sobre la pantalla del dispositivo. Para ello importamos la librería Android.view.MotionEvent, y programamos el correspondiente método asociado, public boolean onTouchEvent(MotionEvent event). Dicho método se llama cada vez que la pantalla táctil del dispositivo es tocada, indicando el pixel de la pantalla (ancho y alto) que ha sido pulsado.

Debido a que OpenGL divide la pantalla en valores que van desde el -1 al 1, tanto en el eje X como en el Y, siendo (0,0) el valor central, deberemos hacer una conversión entre los valores de alto y ancho de la pantalla que nos devuelve el método onTouchEvent y los que utiliza OpenGL.



Para obtener el alto y ancho de la pantalla del dispositivo importamos la librería Android.view.WindowManager, y nos creamos un objeto de la clase Display. Mediante la llamada a los métodos getWidth() y getHeight() obtenemos el ancho y el alto del dispositivo desde el que estamos ejecutando.

En el método onTouchEvent comprobamos, mediante las variables correspondientes, si hay algún robot o algún ítem pintado en la pantalla en ese momento. Si es así, comprobamos en qué lugar exacto de la pantalla está pintado (según la distancia y el ángulo según se ha explicado en capítulos anteriores) y teniendo en cuenta su tamaño (también puesto en función de la distancia)

comprobamos si se está presionando la pantalla táctil en ese mismo sitio. Si es así, activamos el correspondiente efecto de sonido (si los efectos están activados), mostramos el mensaje de aviso, y llamamos al método correspondiente que avisa al otro proyecto.

```
if(pintado==true)
{
  for (int c=0;c<objetos_pintar.size();c++){
      if(angulos.elementAt(c)>330&&angulos.elementAt(c)<360&&
      Jugar.azimuth<30){
        ejeX = (float) ((angulos.elementAt(c)-
        (Jugar.azimuth+360))/30);
      }
      if(angulos.elementAt(c)>0&&angulos.elementAt(c)<30&&
      Jugar.azimuth>330){
        ejeX = (float) ((angulos.elementAt(c)-
        (Jugar.azimuth-360))/30);
      }else
        ejeX = (float) ((angulos.elementAt(c)-Jugar.azimuth)/30);

  if (distancias.get(c) <7 && distancias.get(c)>4){
      if(a>(mitad_ancho + (mitad_ancho*ejeX)-(ancho*0.09))&&
           a<(mitad_ancho + (mitad_ancho*ejeX)+(ancho*0.09)))
      {
         if(b>(mitad_alto+(2*(mitad_alto/10))-(ancho*0.09)) &&
         b<(mitad_alto+(2*(mitad_alto/10))+(ancho*0.09)))
        {
                       //Acciones
        }
      }
  }
```

Código 21. Control eventos táctiles para la captura de un robot que se encuentra a una distancia de entre 4 y 7 metros del usuario.

Siendo:

- a: el valor del ancho de la pantalla donde ha tenido lugar el evento táctil.
- b: valor de la longitud de la pantalla donde ha tenido lugar el evento táctil.
- alto: la longitud de la pantalla del dispositivo.
- ancho: la anchura de la pantalla del dispositivo.
- mitad_alto: la mitad de la altura de la pantalla del dispositivo.
- mitad_ancho: la mitad de la anchura de la pantalla del dispositivo.

Mensajes

Para interactuar con el usuario y poder notificarle cuándo se ha producido la captura de algún robot o alguna ayuda, utilizamos los *Toast* proporcionados por Android. Utilizamos éstos ya que:

- no requieren ningún tipo de confirmación por parte del usuario
- no reciben el foco de actividad
- desaparecen de la pantalla transcurridos unos segundos.

Para personalizar los *toast* les añadimos una imagen y un texto diferente, en función de si se ha capturado un robot o se ha conseguido una ayuda.

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,
                      (ViewGroup)
findViewById(R.id.toast_layout));

ImageView image = (ImageView) layout.findViewById(R.id.t_image);
image.setImageResource(R.drawable.robot);
TextView text = (TextView) layout.findViewById(R.id.t_text);
text.setText("¡ c a p t u r a d o !");
Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_SHORT);
toast.setView(layout);
```

Código22. Aviso objeto capturado

Audio

La clase audio es la clase más simple de toda la implementación. En el constructor creamos los objetos de la clase MediaPlayer, uno para el efecto asignado a la captura de los robots, otro para la captura de los ítems y otro para avisar cuando el tiempo está finalizando. Los audios está almacenados en la carpeta res→raw.

```
public Audio(Context c)
{
      efecto_objeto=MediaPlayer.create(c,R.raw.e_robot);
      efecto_item=MediaPlayer.create(c,R.raw.e_item);
      efecto_tiempo=MediaPlayer.create(c,R.raw.e_tiempo);

}
```

Código 23. Efectos de sonido

Consta de tres métodos: public void audio_objeto(), public void audio_item() y public void audio_tiempo(), que inician la reproducción de los efectos asignados a los robots, a los ítems y al final del tiempo, respectivamente. Si hay otra reproducción en curso, primero pausa dicha reproducción y después inicia la correspondiente.

```
public void audio_objeto()
{
      if(efecto_item.isPlaying())
      {
            efecto_item.pause();
      }
      if(efecto_objeto.isPlaying())
      {
            efecto_objeto.pause();
      }

      efecto_objeto.start();
}

public void audio_item()
{
      if(efecto_objeto.isPlaying())
      {
            efecto_objeto.pause();
      }
      if(efecto_item.isPlaying())
      {
            efecto_item.pause();
      }
      efecto_item.start();
}
```

Código 24. Efectos de sonido

Una vez que tenemos controlados los eventos táctiles sobre la pantalla del dispositivo, creados los mensajes de aviso de capturas, implementados los métodos para la reproducción de los efectos de sonido, y registrado nuestro Observer en el GestorObservaciones, solo queda hacer la llamada correspondiente a cada característica cuando sea necesario.

```
if (distancias.get(c) <7 && distancias.get(c)>4){
      if(a>(mitad_ancho + (mitad_ancho*ejeX)-(ancho*0.09))&&
      a<(mitad_ancho + (mitad_ancho*ejeX)+(ancho*0.09)))
      {
        if(b>(mitad_alto+(2*(mitad_alto/10))-(ancho*0.09)) &&
        b<(mitad_alto+(2*(mitad_alto/10))+(ancho*0.09)))
        {
            toast.show();
            if(Interfaz.activarEfectos==true)
                    audio.audio_objeto();
             gestor.capturaObjeto(objetos_pintar.get(c));
        }
      }
}
```

Código 25. Acciones correspondientes a la captura de un objeto (robot) que se encuentra a una distancia de entre 4 y 7 metros del usuario.

En el código anterior, cuando se produce la captura de un objeto (robot) que se encuentra a una distancia menor de 7 metros y mayor de 4 respecto a la posición del usuario:

1. Se muestra el mensaje de aviso de captura correspondiente (`toast.show()`).
2. Si los efectos de sonido están activados (`if(Interfaz.activarEfectos==true)`) se reproduce el sonido correspondiente a la captura del objeto.
3. Por último se hace una llamada al método *capturaObjeto*(Integer numero) de nuestro GestorObservaciones, para que notifique a todos los observadores registrados que se ha producido la captura del objeto cuya posición en el array viene determinada por `(objetos_pintar.get(c))`.

3.4. Pruebas

Para comprobar la viabilidad del proyecto, y asegurar una integración correcta con otros sistemas para la realización de videojuegos u otras aplicaciones similares, es necesario garantizar la funcionalidad de éste. Para ello creamos un interfaz de prueba en la que el usuario puede introducir las coordenadas de donde se encuentre y las coordenadas de la posición donde desea encontrar al objeto.

Una curiosidad del sistema Android es que nos ofrece la posibilidad de crear las interfaces gráficas a través ficheros XML, lo que nos permite separar la funcionalidad de las clases y la estética. De esta forma nuestro código queda mucho más ordenado y claro, indicando en los archivos XML la distribución de nuestros objetos gráficos (botones, pestañas…), así como su color, forma, tamaño, colocación etc., mientras que en los archivos java podemos centrarnos en la funcionalidad de éstos y su integración en el sistema.

Por tanto mediante un fichero XML definimos los objetos gráficos que van a comprender la interfaz de interacción con el usuario así como sus diferentes características, quedando la siguiente interfaz gráfica:

Imagen 52. Interfaz de prueba.

En ella el usuario debe introducir los valores de grados, minutos y segundos, norte o sur, oeste o este, de la longitud y de la latitud tanto de su posición como de la posición ficticia del objeto a encontrar.

Imagen 53. Elección valor coordenada E-O.

Imagen 54. Elección valor coordenada N-S.

En primer lugar se comprueba que los valores introducidos de dichas coordenadas sean válidos. Para ello diseñamos un algoritmo que compruebe que el valor introducido en grados de la longitud sea mayor que 0 y menor que 180, y que el de los minutos y segundos sea mayor que 0 y menor que 60, teniendo en cuenta que el valor de los segundos puede ser decimal. El valor de 180 para los grados sería válido en caso de que los minutos y segundos tuvieran un valor de 0. De la misma manera se procede con los valores de la latitud, pero ésta vez comprobando que los grados no tengan un valor mayor de 90. En caso de que el usuario introduzca el valor de alguna coordenada no válido, el sistema le indicará cuál de éstas tiene un valor incorrecto.

Imagen 55. Aviso coordenada no válida.

Una vez que las coordenadas son válidas se inicia la clase Jugar y se añaden las coordenadas del usuario mediante el método mencionado anteriormente, *actualizaPosicion,* y las coordenadas del objeto mediante la clase Elemento, también explicada anteriormente.

```
Jugar.actualizaPosicion(longitudUsuario,   latitudUsuario);

Elemento elemento = new Elemento();
elemento.setLongitud(longitudObjeto);
elemento.setLatitud(latitudObjeto);
elemento.setVisible(true);
elemento.aniadir();

Intent i1 = new Intent (this, Jugar.class);
startActivity(i1);
```

Código25.Añadir coordenadas de usuario y objeto

Posteriormente se comprueba que la distancia entre las coordenadas del usuario y las del objeto a encontrar no disten en más de 50 metros, para que la simulación del sistema no se alargue en demasía.

```
public void comprobarDistancia()
{
      double dis =calcular_distancia((objetos.elementAt(0).
            getLongitud())/ 1E6,
                  (objetos.elementAt(0).getLatitud())/1E6);
      if (dis>50)
      {
            Toast.makeText(this, "Demasiada distancia hasta
                        el objeto", 1000).show();
            finish();
      }

}
```

Código26.Comprobar la validez de la distancia

Para simular un acercamiento del usuario al objeto creamos un hilo y, mediante las fórmulas del rumbo loxodrómico explicadas en el apartado anterior, vamos modificando las coordenadas del usuario a una velocidad constante hacia la posición en la que se encuentra el objeto, de tal manera que se visualice a éste por la pantalla del dispositivo y se pueda observar el ficticio acercamiento del usuario.

```
protected void miHilo ()
{
      Thread t = new Thread ()
      {
            public void run ()
            {
                  while(true)
                  {
                        try
                        {
                              Thread.sleep(10000);
                        }
                        catch (InterruptedException e)
```

```
                                {
                                      e.printStackTrace();
                                }
                                mHandler.post(ejecutarAccion);
                          }
                    }
              };
              t.start();
        }
```

Código27.Hilo que simula movimiento del usuario

Para ello, en primer lugar obtenemos el valor del ángulo en el que se encuentra el objeto con respecto al norte, y la distancia que queremos que se mueva el usuario utilizando una velocidad constante. Con estos dos valores, la última posición del usuario y haciendo uso de las fórmulas vamos calculando la nueva longitud y la nueva latitud de dicho usuario, hasta que se encuentra a menos de 2 metros del objeto.

```
final Runnable ejecutarAccion = new Runnable ()
{
  public void run(){
      double pDis =
      calcular_distancia((objetos.elementAt(0).getLongitud())
      /1E6, (objetos.elementAt(0).getLatitud())/1E6);
      if(pDis>2){
       double R = dameAngulo((objetos.elementAt(0).getLongitud())
       /1E6,(objetos.elementAt(0).getLatitud())/1E6);
         double R2=Math.toRadians(R);
         double velocidad = 0.65;
         double tiempo =3;
         double D =(velocidad * tiempo)/(1852*60);

         double longitud_final;
         double latitud_final;
         longitud_final = (D * Math.sin(R2)/C) + long_usu;
         latitud_final = ((longitud_final - long_usu)*
           C/Math.tan(R2))+ lat_usu;
         actualizaPosicion(longitud_final, latitud_final);
      }
    }
};
```

Código 28.Método que calcula las nuevas posiciones del movimiento del usuario

Haciendo uso del Observable creado para notificar la captura de un objeto creamos la clase PruebaObserver, en la que se elimine el objeto capturado cuando se lleve a cabo la notificación.

```
public class PruebaObserver implements Observer
{
	@Override
	public void update(Observable arg0, Object arg1)
	{
		Jugar.objetos.remove(0);
	}
}
```

Código 29. Clase PruebaObserver

Y registramos dicho Observer en el GestorObservaciones.

```
gestor = new GestorObservaciones();
PruebaObserver po = new PruebaObserver();
gestor.addObserver(po);
```

Código 30.Registro del Observer en el GestorObservaciones

Para poder disponer de las opciones de audio que ofrece la aplicación, éstas se deben activar o desactivar en el menú principal, mediante un *AlertDialog*:

```
private void abrirOpcionesAudio(){
	new AlertDialog.Builder(this).
	setTitle(R.string.opciones_audio_tittle).
	setItems(R.array.audio, new
			DialogInterface.OnClickListener()
	  {

	@Override
	public void onClick(DialogInterface dialog, int
			which) {
			if(which==0){
				activarEfectos=true;
			}
			else{
				activarEfectos=false;
			}
	}}).show();
}
```

Código 31.Activación/desactivación de los efectos de sonido

Si el usuario no hace uso de las opciones de audio, por defecto los efectos de sonido estarán activados.

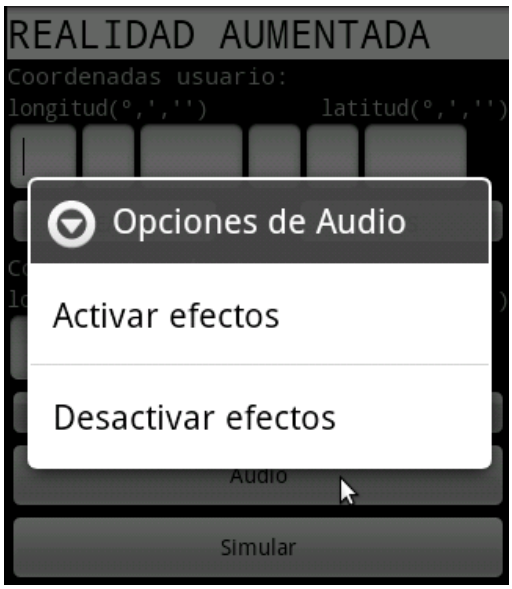


Imagen 56. Opciones de audio.

Capítulo 4.

Conclusiones

Para obtener las conclusiones de un proyecto software como éste, basta con analizar si los objetivos planteados desde un primer momento, ver si éstos han sido cumplidos y en que grado se ha hecho.

Recordemos los objetivos marcados al inicio del proyecto:

1. Conocer las características que ofrecen los teléfonos móviles y como éstas pueden ser aprovechadas para la creación de un sistema de realidad aumentada.

2. Conocer la plataforma Android.

3. Estudiar la realidad aumentada, qué es, y qué aplicaciones tiene.

4. Conocer el estándar OpenGL, las opciones gráficas que ofrece, y como integrarlas en un proyecto.

5. Enfocar diferentes puntos del mismo para que puedan ser utilizados en el videojuego *Invasión Androide*.

1. **Conocer las características que ofrecen los teléfonos móviles y como éstas pueden ser aprovechadas para la creación de un sistema de realidad aumentada:** el principal objetivo de este proyecto consistía en conocer las características que ofrecen hoy en día los llamados *smartphones* y como éstas podían ser explotadas para la creación del sistema de realidad aumentada. Se han estudiado en profundidad los sensores del teléfono, concretamente el sensor de orientación, y podemos afirmar que el objetivo de utilizarlos para dar una mayor sensación de realidad al sistema se ha cumplido. Prueba de ello, es que tanto la posición de los objetos sobre la pantalla, la decisión de mostrar o no mostrar dichos objetos, como su movimiento hacen uso de éstos sensores, abarcando por tanto la parte más fundamental del proyecto. Sin la inclusión de esta tecnología hubiera sido imposible poder realizar el sistema, puesto que no se tendría ninguna conciencia de la posición del dispositivo móvil, así como de su orientación. También se hace uso de la cámara del dispositivo, pues el sistema debe acceder a la imagen proporcionada por ésta constantemente y mezclarla con los objetos diseñados. Este punto también queda cubierto satisfactoriamente.

2. **Conocer la plataforma Android:** otro de los objetivos fundamentales del proyecto consistía en obtener el suficiente conocimiento de la plataforma Android para poder cumplir el resto de objetivos. El primer paso para realizar una aplicación para un determinado sistema, consiste en conocer éste con la suficiente profundidad como para saber si el proyecto es viable y hasta qué grado van a poder cumplirse los objetivos marcados. Podemos afirmar que esto

también se ha cumplido, ya que el sistema puede integrarse para llevar a cabo cualquier aplicación Android con versión mínima 2.1.

3. **Estudiar la realidad aumentada, qué es, y qué aplicaciones tiene:** el profundo estudio que se ha llevado a cabo sobre la realidad aumentada y las aplicaciones que ésta tiene, viendo como se ha ido introduciendo en campos como la educación, la publicidad, el turismo o la medicina entre otros, ha permitido cubrir con éxito este objetivo y sobre todo poder realizar un innovador sistema en constante crecimiento para aplicaciones móviles que ocupan cada día una mayor parte del mercado de la telefonía móvil.

4. **Conocer el estándar OpenGL, las opciones gráficas que ofrece, y como integrarlas en un proyecto:** otra de las partes que también han llevado un estudio ha sido el estándar OpenGL, del cual no se había estudiado nada durante los cursos de carrera. Los objetivos de conocer dicho estándar, las opciones que ofrece y sobre todo como poder integrar los objetos construidos en una aplicación Android han sido cubiertos satisfactoriamente. Si bien cabe señalar que uno de los principales problemas que ha habido a la hora de desarrollar el proyecto y que como se puede ver en el anexo *A. Planificación* de la presente memoria ha hecho que las fechas programadas se retrasaran ha sido integrar los objetos de OpenGL con la imagen de la cámara utilizando las correspondientes transparencias, finalmente han sido diseñados tanto los robots como los ítems con dicho estándar y siendo estos perfectamente visualizados mediante la pantalla del dispositivo junto con la imagen de la cámara capturada en cada instante.

5. **Enfocar diferentes puntos del mismo para que puedan ser utilizados en el videojuego *Invasión Androide*:** como bien se ha explicado durante el transcurso de la memoria, el sistema de realidad aumentada está diseñado para que pueda integrase a la hora de construir diferentes tipos de aplicaciones, sin embargo su principal cometido son los videojuegos. Prueba de ello es la inclusión de la opción de poder elegir el utilizar hasta tres diferentes tipos de ítems, pudiendo dotar así a los videojuegos de mayor dinámica y opciones de entretenimiento. Este objetivo también ha sido cumplido satisfactoriamente pudiendo comprobarse en el videojuego *Invasión Androide*, el cual utiliza el presente sistema de realidad aumentada para su creación, habiéndose integrado perfectamente todos los módulos del sistema (sensores, imagen de la cámara y gráficos, efectos de sonido, eventos de pantalla) con los módulos necesarios para la obtención de un videojuego completo basado en un sistema de realidad aumentada. Por ello los objetos diseñados, tanto los robots, como los ítems, corresponden a los elementos que dicho videojuego *Invasión Androide* necesitaba. Este objetivo puede considerarse como doble, habiéndose realizado satisfactoriamente el trabajo en grupo que conlleva la realización del proyecto, teniendo que cubrir las necesidades del videojuego *Invasión Androide* y por ello coordinar en determinados momentos el desarrollo de ambos proyectos.

Tanto los objetivos principales del proyecto expuestos en este apartado como los diferentes requisitos que se han planteado a lo largo de la memoria (visualización de los objetos en función de su distancia y orientación, avisos sonoros y visuales sobre la captura de los objetos, tiempos de respuesta…), han sido cubiertos satisfactoriamente, por lo que podemos concluir que la realización del sistema se ha llevado a cabo con éxito.

Capítulo 5.

Trabajos Futuros

Aunque los objetivos marcados han sido cumplidos con éxito, existen varias líneas futuras que pueden llevarse a cabo para mejorar y darle continuidad a este pequeño sistema:

1. Mejorar el diseño gráfico de los objetos.
2. Mejorar la captura de los objetos sobre el dispositivo.
3. Mejorar algunos aspectos sobre la visualización de los objetos sobre la pantalla del dispositivo.
4. Posibilidad de utilizar varios ítems a la vez.
5. Eliminar el efecto de refresco de la pantalla.

1. **Mejorar el diseño gráfico de los objetos:** El primero de ellos trataría de mejorar el diseño de los elementos incluidos. Como se mencionó en el capítulo de desarrollo, en un primer lugar se pensó diseñar ratones en vez de robots para ser capturados, sin embargo el total desconocimiento en un primer lugar del estándar OpenGL y la complejidad que supuso poder diseñar esferas en tres dimensiones que dieran sensación de realidad a lo hora de utilizar los diferentes sensores de movimiento del dispositivo, hizo que hubiera que optar por crear objetos a partir de polígonos. Por tanto, este proyecto deja abierta la posibilidad de diseñar elementos gráficos complejos, que doten al sistema de una mayor sensación de realidad y pueda este integrarse para construir aplicaciones que necesiten objetos más reales y de contornos más complejos que polígonos regulares.

2. **Mejorar la captura de los objetos sobre el dispositivo:** Si bien es verdad que sería prácticamente inapreciable por el usuario, se podría incluir como mejora el sistema de captura de los objetos sobre la pantalla. En el presente proyecto, se ha hecho uso de las dimensiones de la pantalla del dispositivo móvil en el que se utilice la aplicación y de los eventos táctiles del teclado, pues conociendo el tamaño de los objetos y la posición de éstos sobre la pantalla, determinada por su distancia al usuario, es fácil calcular en qué posición exacta de la pantalla se encuentra. Sin embargo el estándar OpenGL permite conocer cuando un objeto ha sido pulsado, independientemente de su posición, y aunque no sea necesario para el presente proyecto debido a las características de éste, podría ser interesante realizar un estudio de cómo obtener la información de la captura de los objetos, sin necesidad de conocer el tamaño y la posición de éstos sobre la pantalla.

3. **Mejorar algunos aspectos sobre la visualización de los objetos sobre la pantalla del dispositivo:** Sin duda, el punto mejor cubierto en el proyecto, ha sido el trabajo con las coordenadas del usuario y los objetos junto con los sensores del teléfono, permitiendo obtener sin error alguno la posición de éstos, su posible visibilidad a través de la pantalla atendiendo a la orientación de éste, su tamaño y localización sobre el dispositivo y la rotación de los mismos. Sin embargo, podrían mejorarse algunos aspectos como:

- Determinar si el objeto se está pintando sobre una acera o un lugar transitable, o por el contrario se ha ubicado en un sitio inadecuado.
- Cuando el usuario acerque su dispositivo colocando la cámara encima del supuesto objeto, que éste visualice dicho objeto desde arriba, es decir, que aunque los sensores del dispositivo lo ubiquen fuera del ángulo de visión, sea posible su correspondiente visualización.

4. **Posibilidad de utilizar varios ítems a la vez:** el sistema permite la visualización simultánea de tantos elementos principales (en este caso robots) como sean necesarios y de un ítem. Sin embargo, como mejora del sistema se podría añadir la posibilidad de utilizar a la vez tantos ítems como se desee, sin la necesidad de tener que capturar uno para poder añadir otro nuevo. El sistema permite el uso de tres ítems diferentes, pero podrían añadirse más si así lo requiriese la aplicación.

5. **Eliminar el efecto de refresco de la pantalla:** la parte en el desarrollo del sistema que más problema trajo consigo, fue la de conseguir que se observaran perfectamente los objetos OpenGL con la imagen de la cámara de fondo. Finalmente, mediante el uso de transparencias en los objetos, y utilizando imágenes .bmp o .jgp para las texturas en vez de los colores proporcionados por OpenGL, se consiguió que éstos pudieran verse delante de la imagen capturada por la cámara. Sin embargo, al tener que añadir estos delante de la imagen de la cámara cuando tuvieran que ser observados, se notan los refrescos de la pantalla, cuando se pasa de no ver ningún objeto a ver alguno, o viceversa. Un trabajo futuro que deja abierto el desarrollo del sistema sería solucionar este punto.

Bibliografía.

- Programación Android. Pedro Dans Álvarez, traductor. Ed Burnette. Anaya Multimedia.

- G. Fernández Arnedo, J.J. Toledano Mancheño. Navegación General y Radionavegación. Editorial AVA.

- Vladimir Silva. Pro Android Games. Editorial Apress.

- http://developer.android.com/

Referencias.

[1] Wikipedia, "Realidad Aumentada" < http://es.wikipedia.org/wiki/Realidad_aumentada> [Último acceso Agosto 2011]

[2] Alt1040, "*Layar*" < *http://alt1040.com/2010/01/las-5-mejores-aplicaciones-de-realidad-aumentada-para-celulares*> [Último acceso Agosto 2011]

[3] What´s New, "*TAT augmented ID*" <*http://wwwhatsnew.com/2011/03/22/%C2%BFque-es-la-realidad-aumentada-evolucion-y-complejidad-de-las-mejores-aplicaciones/* > [Último acceso Agosto 2011]

[4] Alt1040, " *Yelp Monocle*" < *http://alt1040.com/2010/01/las-5-mejores-aplicaciones-de-realidad-aumentada-para-celulares*> [Último acceso Agosto 2011]

[5] The Guardian, "Global games market to be worth $74bn in 2011" <http://www.guardian.co.uk/technology/gamesblog/2011/jul/06/gartner-global-game-spending> [Último acceso Agosto 2011]

[6]Wikipedia, "iPhone" < http://es.wikipedia.org/wiki/IPhone> [Último acceso Agosto 2011]

[7] Wikipedia, "Android" <http://es.wikipedia.org/wiki/Android> [Último acceso Agosto 2011]

[8] Android Market, <https://market.android.com/?hl=es> [Último acceso Agosto 2011]

[9] Conecttados, "Ya se descargan más aplicaciones de Android que de iOS"

< http://www.conecttados.com/23498-ya-se-descargan-mas-aplicaciones-de-android-que-de-ios > [Último acceso Agosto 2011]

[10] Wikipedia, "GPS" <http://en.wikipedia.org/wiki/Global_Positioning_System> [Último acceso Agosto 2011]

[11] Wikipedia, "OpenGL" <http://es.wikipedia.org/wiki/OpenGL> [Último acceso Julio 2011]

[12] Wikipedia, "Realidad virtual" < http://es.wikipedia.org/wiki/Realidad_virtual> [Último acceso Agosto 2011]

[13] Madrimasd, "Realidad Aumentada" <http://www.madrimasd.org/blogs/sistemas_inteligentes/2008/03/21/87063> [Último acceso Agosto 2011]

[14] El blog de Asturel, "Realidad Aumentada" <http://asturel.blogcindario.com/2010/03/00811-realidad-aumentada.html> [Último acceso Agosto 2011]

[15] Leedsmet, "First IEEE International Workshop on Augmented Reality"

<http://augmented-reality.inn.leedsmet.ac.uk/iwar/98/> [Último acceso Agosto 2011]

[16] Wearable Computer Lab, "ARQuake: Interactive Outdoor Augmented Reality Collaboration System" <http://wearables.unisa.edu.au/projects/arquake/> [Último acceso Agosto 2011]

[17] Celulares, "Wikitude AR Guia de viaje" < http://celulares.cr/realidad-aumentada-software-para-celulares/ > [Último acceso Agosto 2011]

[18] Augmented Reality Logo, <http://augmentedrealitylogo.com/> [Último acceso Agosto 2011]

[19]Genbeta, "Metaido trae la realidad aumentada a los libros" <http://www.genbeta.com/actualidad/metaio-trae-la-realidad-aumentada-a-los-libros > [Último acceso Agosto 2011]

[20] MMM, “AR-Rehab (Realidad Aumentada y rehabilitación)” <http://www.mariommoreno.com/blog/2010/12/ar-rehab-realidad-aumentada-y-rehabilitacion/> [Último acceso Agosto 2011]

[21] 7PMix, “Realidad Aumentada para FIAT 500” <http://www.7pmix.com/tag/promocion/page/3/> [Último acceso Agosto 2011]

[22] Zugara, <http://zugara.com/> [Último acceso Agosto 2011]

[23] Tissot, <http://www.tissot.ch/>[Último acceso Agosto 2011]

[24] No Solo Unix, “Wikitude World Browser para Android” <http://www.nosolounix.com/2011/05/realidad-aumentada-android.html> [Último acceso Agosto 2011]

[25] El País, “Realidad Aumentada para arreglar el motor de un coche” <http://www.elpais.com/articulo/tecnologia/Realidad/aumentada/arreglar/motor/coche/elpeputec/20110331elpeputec_4/Tes> [Último acceso Agosto 2011]

[26] Configurar Equipos, “Qué es la realidad aumentada y sus aplicaciones“ <http://www.configurarequipos.com/doc1214.html> [Último acceso Agosto 2011]

[27] Indice Latino, “Historia de los videojuegos: Orígenes” <http://indicelatino.com/juegos/historia/origenes/> [Último acceso Agosto 2011]

[28] Wikipedia, “Atari” <http://es.wikipedia.org/wiki/Atari> [Último acceso Agosto 2011]

[29] Pong Story, “Magnavox Odyssey” <http://www.pong-story.com/odyssey.htm> [Último acceso Agosto 2011]

[30] Nintendo, “La historia de Nintendo” <http://www.nintendo.es/NOE/es_ES/service/la_historia_de_nintendo_9911.html> [Último acceso Agosto 2011]

[31]Sony, “Historia” <http://www.sony.es/hub/careers/3/1> [Último acceso Agosto 2011]

[32] DxMax, “La gran historia de Sega” <http://dxmax.es.tl/La-gran-historia-de-sega.htm> [Último acceso Agosto 2011]

[33] Indice Latino, “Historia de los videojuegos: empresas de videojuegos” <http://indicelatino.com/juegos/historia/empresas/> [Último acceso Agosto 2011]

[34] El otro lado, “Historia de los videojuegos: década de los 70” <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos:_Decada_de_los_70> [Último acceso Agosto 2011]

[35] Pong Story, “Atari Pong” < http://www.pong-story.com/atpong2.htm> [Último acceso Agosto 2011]

[36] El otro lado, “Historia de los videojuegos: década de los 80” <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos:_Decada_de_los_80> [Último acceso Agosto 2011]

[37] Wikipedia, “Super Mario Bros” <http://es.wikipedia.org/wiki/Super_Mario_Bros> [Último acceso Agosto 2011]

[38]Nintendo, “Game Boy” <http://www.nintendo.es/NOE/es_ES/systems/game_boy_1425.html> [Último acceso Agosto 2011]

[39] Wikipedia, “Game & Watch” <http://es.wikipedia.org/wiki/Game_%26_Watch> [Último acceso Agosto 2011]

[40] C64, <http://www.c64.com/> [Último acceso Agosto 2011]

[41] Wikipedia, "Mario Bros" <http://en.wikipedia.org/wiki/Mario_Bros> [Último acceso Agosto 2011]

[42] Tetris, < http://www.tetris.com/ > [Último acceso Agosto 2011]

[43] Wikipedia, "Boberman" <http://es.wikipedia.org/wiki/Bomberman> [Último acceso Agosto 2011]

[44]Museo 8 bits, "Sega", <http://www.museo8bits.com/sega8_ms.htm> [Último acceso Agosto 2011]

[45] El otro lado, "Historia de los videojuegos: década de los 90" <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos:_Decada_de_los_90> [Último acceso Agosto 2011]

[46] Punto de partida, "Super Nintendo" <http://www.puntodepartida.com/retroinformatica/maquinadeltiempo/supernintendo.php> [Último acceso Agosto 2011]

[47] Mundogamers, <http://www.mundogamers.com/foros/2884-historia-de-la-mega-drive.html> [Último acceso Agosto 2011]

[48] Taringa, "Historia de Sonic" <http://www.taringa.net/posts/info/1414408/Historia-de-Sonic.html> [Último acceso Agosto 2011]

[49] Wikipedia, "Street Fighter" <http://es.wikipedia.org/wiki/Street_Fighter> [Último acceso Agosto 2011]

[50] Wikipedia, "FIFA" <http://es.wikipedia.org/wiki/FIFA_(serie) > [Último acceso Agosto 2011]

[51] Dos Ideas, "La historia de Play Station" <http://www.dosideas.com/noticias/entretenimientos/542-la-historia-de-la-playstation.html> [Último acceso Agosto 2011]

[52] Wikipedia, "Final Fantasy" <http://es.wikipedia.org/wiki/Final_Fantasy> [Último acceso Agosto 2011]

[53] Wikipedia, "Grand Theft Auto" <http://es.wikipedia.org/wiki/Grand_Theft_Auto> [Último acceso Agosto 2011]

[54] Gizmologia, "Juegos móviles: de Snake a Infinity Blade" <http://gizmologia.com/2011/07/juegos-moviles-de-snakes-a-infinity-blade> [Último acceso Agosto 2011]

[55] El otro lado, "Historia de los videojuegos: Era moderna" <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos:_Era_moderna> [Último acceso Agosto 2011]

[56] Wikipedia, "Shenmue" <http://es.wikipedia.org/wiki/Shenmue> [Último acceso Agosto 2011]

[57] Wikipedia, "Wifi" <http://es.wikipedia.org/wiki/Wi-Fi> [Último acceso Agosto 2011]

[58] Play Sation, <http://es.playstation.com/ps3/> [Último acceso Agosto 2011]

[59] Nintendo, "Nintendo 3DS" <http://www.nintendo.es/NOE/es_ES/nintendo_3ds_23802.html> [Último acceso Agosto 2011]

[60] Espegizmo, “Realidad Aumentada Layar 3D trae a Pac Man a la vida real” <http://espegizmo.com/2009/09/22/general/realidad-aumentada-layar-3d-trae-a-pac-man-a-la-vida-real/> [Último acceso Agosto 2011]

[61] Acetia, “Ghostwire te hace ver fantasmas” <http://ecetia.com/2009/10/ghostwire-te-hace-ver-fantasmas> [Último acceso Agosto 2011]

[62] Configurar Equipos, “Invizimals, realidad aumentada para PSP” <http://www.configurarequipos.com/actualidad-informatica/1449/invizimals-realidad-aumentada-para-psp> [Último acceso Agosto 2011]

[63] Microsoft, “Kinect para Xbox 360” <http://www.microsoft.com/latam/prensa/2010/junio/KINECT.aspx> [Último acceso Agosto 2011]

[64] MIX, “Eye Pet para Play Station3 y la realidad aumentada en los videojuegos” <http://mix.pe/contenido/eyepet-para-playstation-3-y-la-realidad-aumentada-en-los-videojuegos> [Último acceso Agosto 2011]

[65] What’s new, “Twitt Around” <http://wwwhatsnew.com/2011/03/22/%C2%BFque-es-la-realidad-aumentada-evolucion-y-complejidad-de-las-mejores-aplicaciones/> [Último acceso Agosto 2011]

[66] Atl1040, “Wikitude World Browser” <http://alt1040.com/2010/01/las-5-mejores-aplicaciones-de-realidad-aumentada-para-celulares> [Último acceso Agosto 2011]

[67] Aeromental, “Arhrrrr: Shooter de realidad aumentada con zombies” <http://www.aeromental.com/2009/06/15/arhrrrr-shooter-de-realidad-aumentada-con-zombies/> [Último acceso Agosto 2011]

[68] EsferaIphone, “Kweekies: juego de realidad aumentada” <http://www.esferaiphone.com/juegos/kweekies-juego-de-realidad-aumentada/> [Último acceso Agosto 2011]

[69] Shadow Cities, <http://www.shadowcities.com/> [Último acceso Agosto 2011]

[70] EsferaIphone, “Gigaputt convierte la ciudad en un campo de golf virtual” <http://www.esferaiphone.com/iphone/gigaputt-convierte-la-ciudad-en-un-campo-de-golf-virtual/> [Último acceso Agosto 2011]

[71] AR Defender, <http://www.ardefender.com/> [Último acceso Agosto 2011]

[72] Wikipedia, “Bada” <http://es.wikipedia.org/wiki/Bada> [Último acceso Agosto 2011]

[73] 4Android, “Spectrek, encuentra a los fantasmas en tu calle” <http://4ndroid.com/spectrek-encuentra-a-los-fantasmas-de-tu-calle/> [Último acceso Agosto 2011]

[74] Zombies, Run!, <https://www.zombiesrungame.com/> [Último acceso Agosto 2011]

[75] Androidtapp, “Sky Siege” <http://www.androidtapp.com/sky-siege/> [Último acceso Agosto 2011]

[76] Parallel Kingdom, <http://www.parallelkingdom.com/> [Último acceso Agosto 2011]

Anexos.

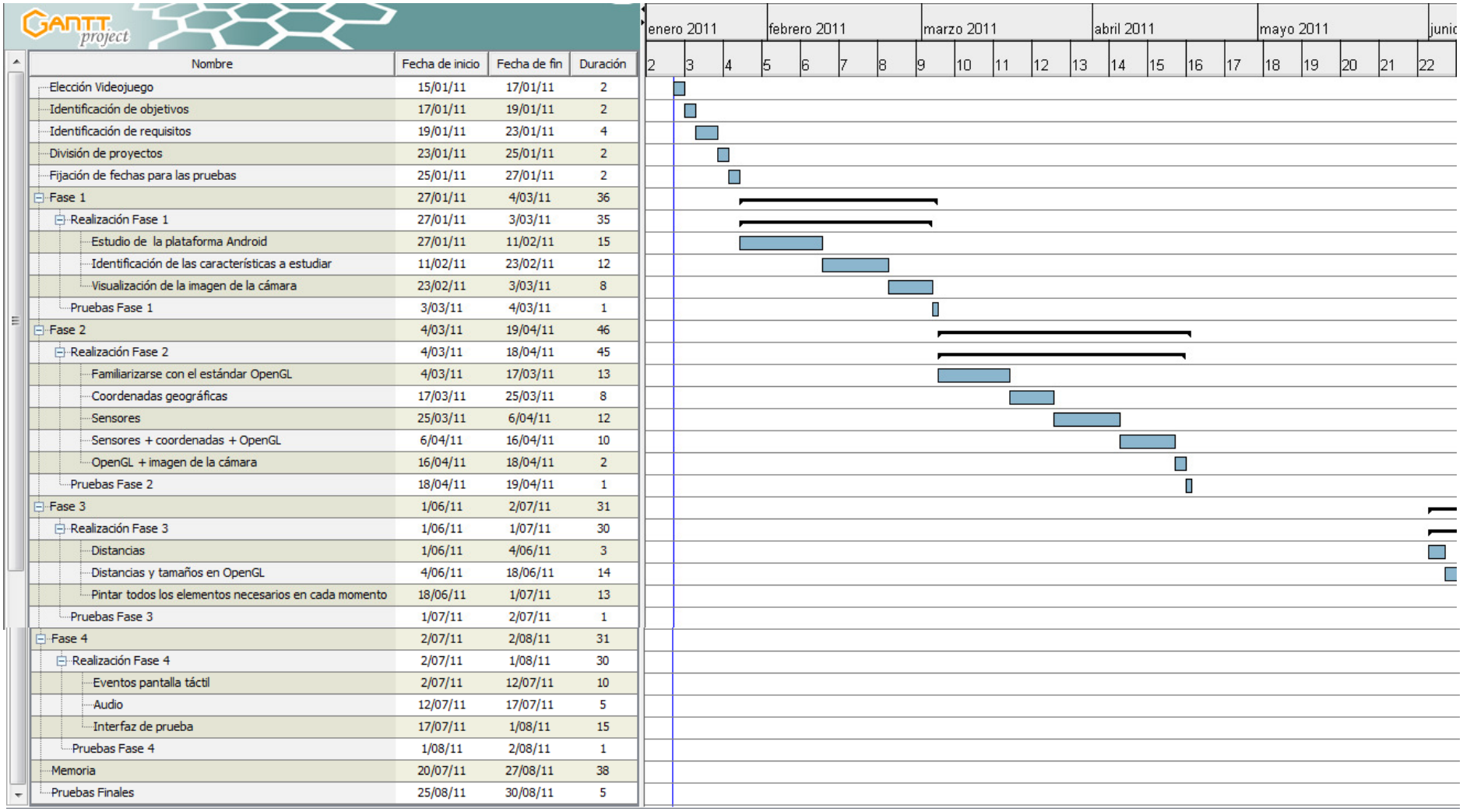
Anexo A. Planificación

La planificación de un proyecto es fundamental para el buen desarrollo de éste. Cumplir con los plazos previstos, saber qué partes son críticas si no se terminan a tiempo y a qué pueden afectar garantizarán el resultado exitoso del mismo.

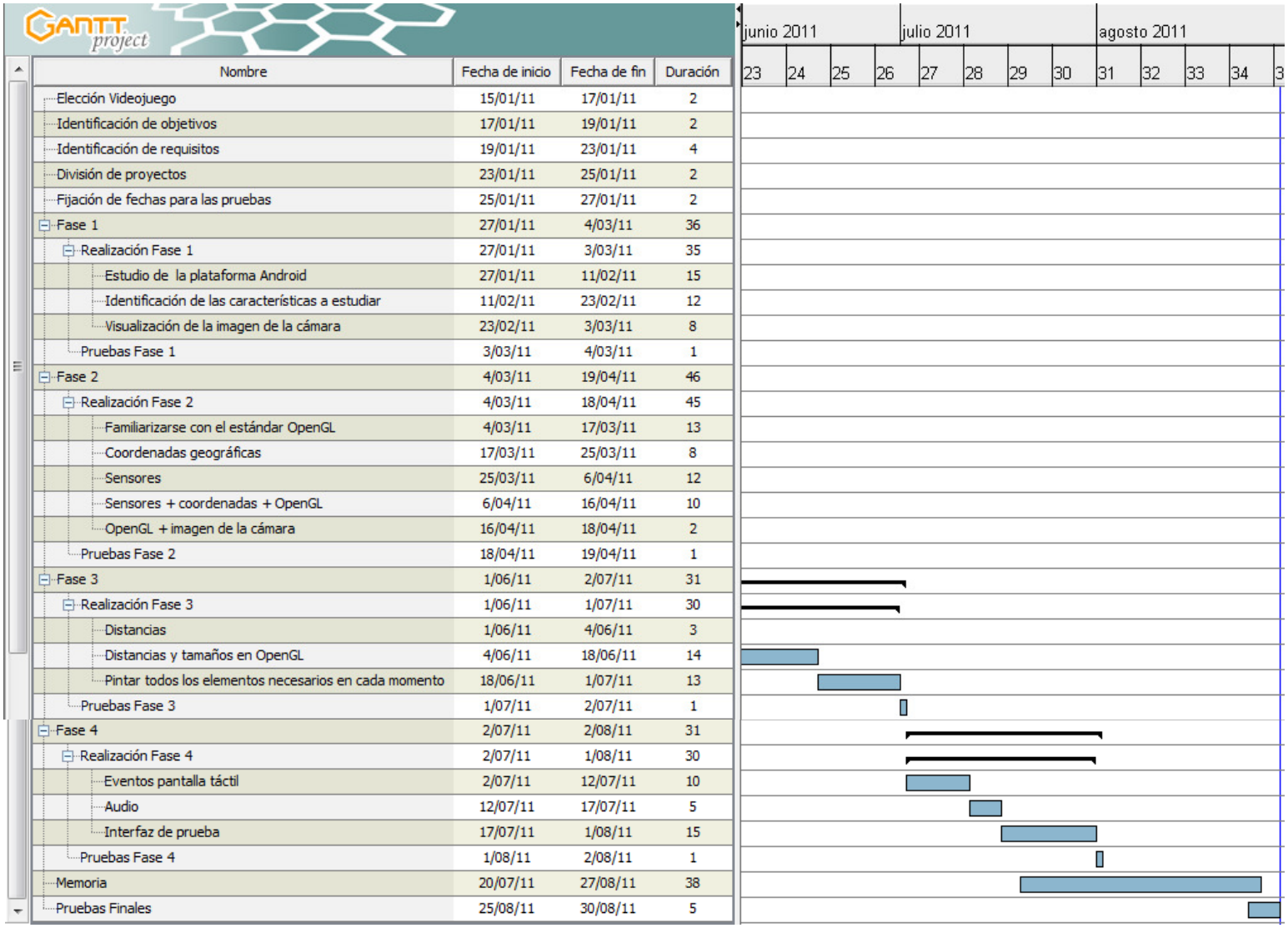
A continuación se presentan las planificaciones iniciales y finales del sistema, para comprobar si se han cumplido los plazos previstos y si no es así que partes han sido las causantes.

A.1. Planificación inicial

Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estudiar	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	19/04/11	46
Realización Fase 2	4/03/11	18/04/11	45
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	16/04/11	18/04/11	2
Pruebas Fase 2	18/04/11	19/04/11	1
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en cada momento	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Pruebas Finales	25/08/11	30/08/11	5



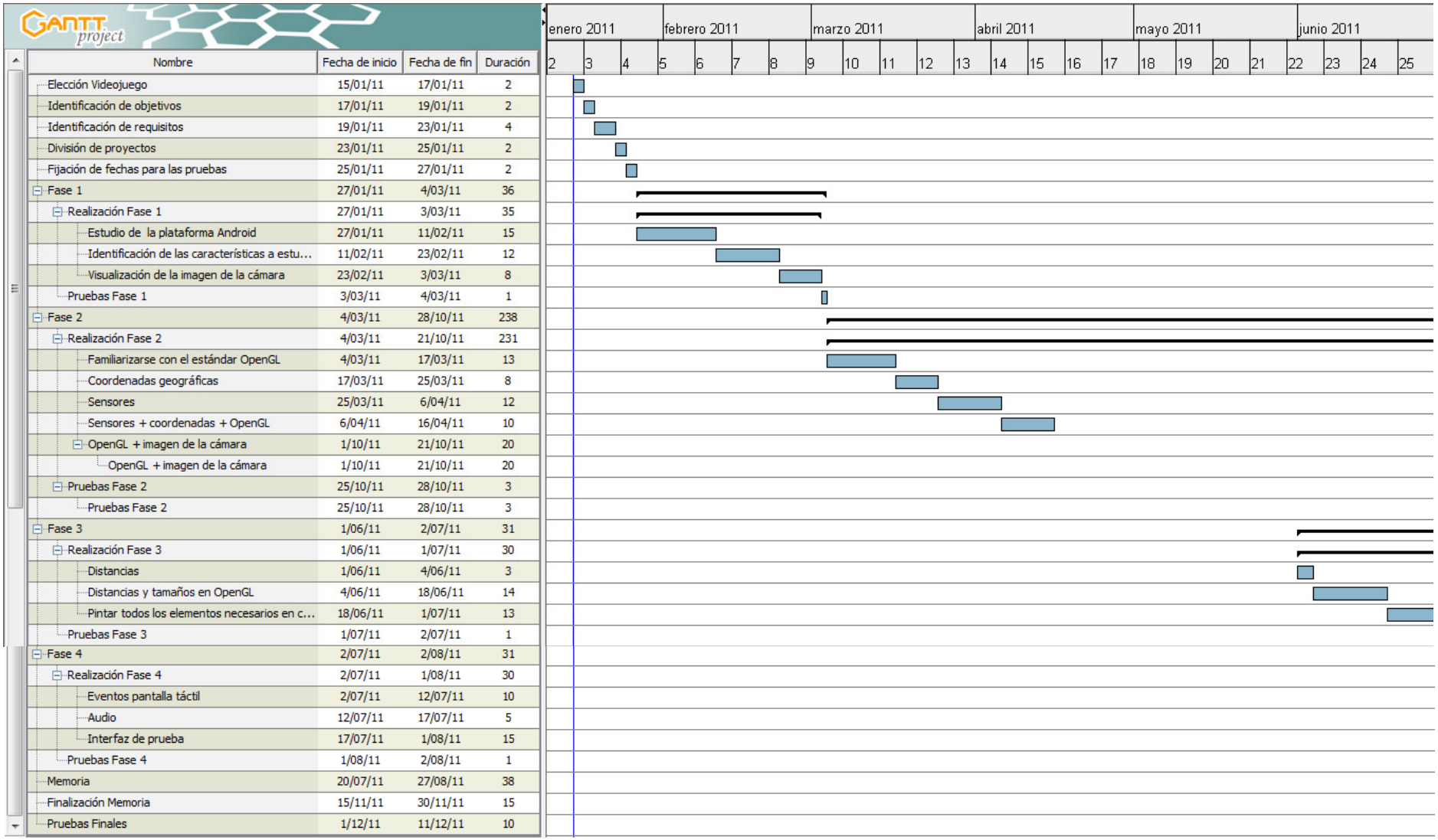
Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estudiar	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	19/04/11	46
Realización Fase 2	4/03/11	18/04/11	45
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	16/04/11	18/04/11	2
Pruebas Fase 2	18/04/11	19/04/11	1
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en cada momento	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Pruebas Finales	25/08/11	30/08/11	5



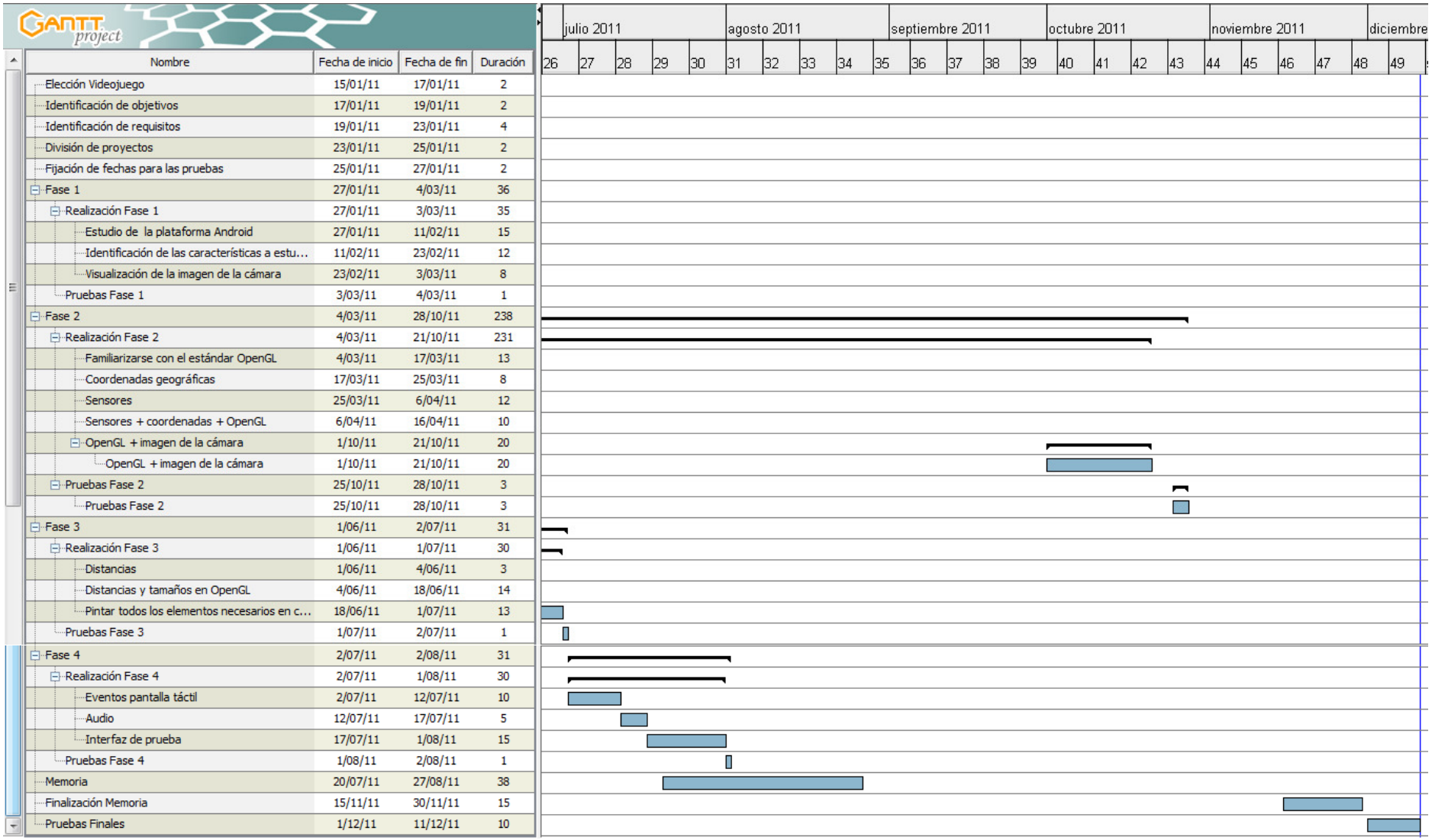
Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estudiar	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	19/04/11	46
Realización Fase 2	4/03/11	18/04/11	45
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	16/04/11	18/04/11	2
Pruebas Fase 2	18/04/11	19/04/11	1
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en cada momento	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Pruebas Finales	25/08/11	30/08/11	5

A2. Planificación final

Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estudiar	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	28/10/11	238
Realización Fase 2	4/03/11	21/10/11	231
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
Pruebas Fase 2	25/10/11	28/10/11	3
Pruebas Fase 2	25/10/11	28/10/11	3
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en cada momento	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Finalización Memoria	15/11/11	30/11/11	15
Pruebas Finales	1/12/11	11/12/11	10



Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estu...	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	28/10/11	238
Realización Fase 2	4/03/11	21/10/11	231
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
Pruebas Fase 2	25/10/11	28/10/11	3
Pruebas Fase 2	25/10/11	28/10/11	3
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en c...	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Finalización Memoria	15/11/11	30/11/11	15
Pruebas Finales	1/12/11	11/12/11	10



Nombre	Fecha de inicio	Fecha de fin	Duración
Elección Videojuego	15/01/11	17/01/11	2
Identificación de objetivos	17/01/11	19/01/11	2
Identificación de requisitos	19/01/11	23/01/11	4
División de proyectos	23/01/11	25/01/11	2
Fijación de fechas para las pruebas	25/01/11	27/01/11	2
Fase 1	27/01/11	4/03/11	36
Realización Fase 1	27/01/11	3/03/11	35
Estudio de la plataforma Android	27/01/11	11/02/11	15
Identificación de las características a estu...	11/02/11	23/02/11	12
Visualización de la imagen de la cámara	23/02/11	3/03/11	8
Pruebas Fase 1	3/03/11	4/03/11	1
Fase 2	4/03/11	28/10/11	238
Realización Fase 2	4/03/11	21/10/11	231
Familiarizarse con el estándar OpenGL	4/03/11	17/03/11	13
Coordenadas geográficas	17/03/11	25/03/11	8
Sensores	25/03/11	6/04/11	12
Sensores + coordenadas + OpenGL	6/04/11	16/04/11	10
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
OpenGL + imagen de la cámara	1/10/11	21/10/11	20
Pruebas Fase 2	25/10/11	28/10/11	3
Pruebas Fase 2	25/10/11	28/10/11	3
Fase 3	1/06/11	2/07/11	31
Realización Fase 3	1/06/11	1/07/11	30
Distancias	1/06/11	4/06/11	3
Distancias y tamaños en OpenGL	4/06/11	18/06/11	14
Pintar todos los elementos necesarios en c...	18/06/11	1/07/11	13
Pruebas Fase 3	1/07/11	2/07/11	1
Fase 4	2/07/11	2/08/11	31
Realización Fase 4	2/07/11	1/08/11	30
Eventos pantalla táctil	2/07/11	12/07/11	10
Audio	12/07/11	17/07/11	5
Interfaz de prueba	17/07/11	1/08/11	15
Pruebas Fase 4	1/08/11	2/08/11	1
Memoria	20/07/11	27/08/11	38
Finalización Memoria	15/11/11	30/11/11	15
Pruebas Finales	1/12/11	11/12/11	10

Como podemos observar en las planificaciones presentadas anteriormente, ha habido un retraso en la Fase 2, en el punto correspondiente a la integración entre los objetos OpenGL y la imagen de la cámara. Esto se debe, como ya se comentó en el apartado de trabajos futuros, a que se tardó más de lo esperado en conseguir que los objetos pudieran observarse en su totalidad teniendo como fondo la imagen de la cámara, lo que ha supuesto un retraso considerable en la planificación del proyecto.

Anexo B. Presupuesto

B1. Presupuesto Sistema de Realidad Aumentada

Atendiendo a la planificación final se puede calcular a cuánto asciende el presupuesto del sistema desarrollado. Se calculará también el presupuesto del videojuego *Invasión Androide* incluyendo la parte del sistema de Realidad Aumentada, para posteriormente poder hacer un estudio de los posibles beneficios que se pueden obtener con una aplicación de este estilo.

La duración del desarrollo del sistema, asciende a 239 días. 20 días corresponden al trabajo realizado por un analista y los 176 restantes al trabajo de programadores. Se puede aproximar que cada día el analista haya dedicado 5 horas de trabajo, y los programadores 3 y media. Por su parte el jefe de proyecto ha dedicado al sistema entero un total de 20 horas.

Cargo Profesional	Retribución (€/año)	Retribución (€/h)	Horas trabajadas	Total (€)
Jefe de proyecto	40.000	22,2	20	444
Analista	30.000	16,6	150	2.490
Programadores	20.000	11,2	731	8.187,2

Por tanto, la realización del sistema supone un coste de 11121,2€.

B2. Presupuesto Videojuego completo

Por su parte, el desarrollo necesario para su integración con el sistema de Realidad Aumentada que conlleva la creación del videojuego *Invasión Androide* asciende a un total de 324 días. 45 días corresponden al trabajo realizado por un analista y los 279 restantes al trabajo de programadores. Se puede aproximar que cada día el analista haya dedicado 5 horas de trabajo, y los programadores 3 y media. Por su parte el jefe de proyecto ha dedicado al sistema entero un total de 25 horas.

Cargo Profesional	Retribución (€/año)	Retribución (€/h)	Horas trabajadas	Total (€)
Jefe de proyecto	40.000	22,2	25	555
Analista	30.000	16,6	225	3.735
Programadores	20.000	11,2	976	10.931,2

Lo que asciende a un total de 26.342,4€ para la elaboración del videojuego.

Se debe hacer un estudio de los posibles beneficios que se pueden obtener con la elaboración del videojuego. Al tratarse de un videojuego Android, la manera de ofrecerlo al público es a través de la aplicación Android Market. Para obtener beneficios habría que subir la aplicación con un precio por descarga. Teniendo en cuenta que los precios de este tipo de videojuegos, siempre que no son gratuitos, son bastante bajos, un precio no descabellado podría ser de 60 céntimos. Para amortizar el coste, habría que tener como mínimo 43.904 descargas, lo que supondría en un año a una media de 3.659 descargas mensuales.

Sin embargo hay que apuntar, que como bien se expuso en el capítulo 2 de esta memoria, hay varias aplicaciones de realidad aumentada con características similares a este proyecto que aparecen de forma gratuita para descargar desde el Android Market. Por tanto, es de esperar, que no se produzcan las descargas esperadas y que se tarde más de un año en amortizar el videojuego, si es que esto llega a producirse.

Una opción más viable para sacar beneficios del videojuego, sería buscar una empresa que quisiera publicitarse en él, que cubriera con los gastos que ha supuesto su elaboración más una cantidad establecida por descarga. De esta manera, se podría establecer la descarga del videojuego gratuitamente y con ello garantizar que el número de descargas seguramente será mucho mayor que fijándole un precio.

Anexo C. Descripción del videojuego *Invasión Androide*

El gobierno había encontrado la manera de realizar importantes estudios sociológicos implicándose de una manera bastante activa en estos. Había diseñado una inteligencia artificial lo suficientemente potente como para que, integrándolo en un prototipo humanoide, e induciéndolo algunos ideales, pudiera integrarse en los grupos sociales más radicales y peligrosos para observar y anotar conductas sociológicas, sin necesidad que esto conllevara un peligro.

En algún punto de uno de los estudios algo salió mal y ahora un grupo de androides con tendencias psicópatas deambula por las diferentes ciudades del mundo, que se encuentran vulnerables ante el desconocimiento de esta nueva amenaza.

El usuario se convierte en un agente secreto del gobierno al que se ha prometido un retiro de por vida sin ningún tipo de preocupaciones si logra, a toda costa, evitar que el fallo del gobierno salga a la luz. Para ello se le ha transportado al centro neurálgico en el que ocurrieron los hechos, será el lugar donde el usuario inicie el juego, y tendrá que identificar y capturar cuantos androides pueda antes de que alguno de ellos cometa su primer crimen, obligando al gobierno a publicar los sucesos e intentar enmendarlos con soluciones más drásticas.

Por suerte los movimientos de los androides están vigilados por satélites y nuestro agente cuenta con un dispositivo, el único en el mundo, que permite diferenciar a estos androides de personas corrientes, y además localizarlos en un mapa. Tendrá que hacer uso de su pericia y rapidez para intentar retrasar lo máximo un hecho imposible de ocultar.

Resulta un juego bastante dinámico en el que el estrés de una cuenta atrás precipita todos los sucesos, acelerando las acciones del usuario. Además con este videojuego se ha querido hacer un guiño al sistema operativo Android, que en sus inicios hacía lo mismo con la película *Blade Runner* con su teléfono *Nexus One*. Es más nuestro juego guarda muchas similitudes con esta película en la que Rick Deckard, un ex agente retirado de los Blade Runners, tiene que encontrar y acabar con unos replicantes fugitivos del modelo *Nexus 6*, seres fabricados a través de la ingeniería genética usados como esclavos en las colonias exteriores de la Tierra para realizar los trabajos más peligrosos.

Dejando de lado todas las peculiaridades del juego vamos a dar paso a una explicación más completa de lo que el videojuego hace y como atiende los diferentes requisitos impuestos.

Hasta ahora sabemos que el objetivo del juego es atrapar unos androides en un tiempo limitado. Estos androides serán un diseño en 3D, creado con la librería que el otro proyecto nos facilita, que más adelante explicaremos dónde, cuándo y por qué se podrán visualizar en el videojuego.

Básicamente se trata de conseguir atrapar, antes de que el tiempo finalice, cuantos más androides posibles mejor, ya que aumentarán la puntación final obtenida. El tiempo con el que cuenta el usuario son 15 minutos, prorrogables siempre y cuando se den una serie de circunstancias, como, por ejemplo, que coja un ítem que para el tiempo. La cuenta atrás estará siempre visible en la pantalla del dispositivo.

Si el contador de tiempo llega a cero sin haber capturado a todos los androides, el jugador habrá fracasado en su misión y será el final del juego. No se habrá logrado ocultar el error del gobierno y el ansiado retiro tras un último trabajo bien hecho tendrá que posponerse a que surja una nueva oportunidad, lo que se tarde en iniciar una nueva partida.

El juego constará de una serie de rondas, en las que habrá un número variable de androides, que crecerá con el paso de los sucesivos niveles. Evidentemente para pasar de ronda tendrás que capturar a todos los androides de ese nivel. Para capturar cada uno de los androides será necesario encontrarlo a través de la cámara del dispositivo y tocarlo en la pantalla del terminal, siempre que la distancia sea apropiada. Ya que solo se podrá identificar a un androide si te encuentras a una distancia menor de 20 metros, apareciendo en la pantalla del teléfono móvil, y sólo podrás capturarlo si te encuentras a menos de 7 metros.

Capturar a un androide supone dos cosas. La primera que éste desaparezca y que por tanto quede un androide menos para pasar de ronda, a no ser que fuera el último, lo que supondrá el cambio automático de ronda. Y la segunda que se incremente el acumulado de puntos conseguidos, un extra del juego que refleja lo bien que se puede hacer y que permite compararlo con puntaciones obtenidas en partidas anteriores.

Si se logra pasar con éxito cada una de las rondas que van surgiendo, que ascienden a un total de diez, el jugador habrá logrado terminar el juego, lo que además de suponer una bonificación extra de puntos para la puntuación final que quedará reflejada en un ranking, conseguirá ese ansiado retiro que el gobierno le había prometido.

Evidentemente los androides no esperarán en un sitio fijo a que les captures. Estos se estarán moviendo de un lado a otro como transeúntes normales, lo que dificultará un poco más la misión del usuario de este videojuego.

En principio se pensó en dejar el número de rondas abierto y que el usuario pudiera disfrutar de niveles elevados donde el dinamismo del juego rozará lo excéntrico, pero finalmente se decidió darle un límite de rondas para que el juego pudiera tener un final feliz.

Con el paso de cada nivel se incrementará el número de androides que capturar, por lo que la dificultad del juego va aumentando. Pero con la sucesión de rondas existe una bonificación de tiempo extra por nivel completado, de tal manera que el tiempo que se tiene para capturar a los androides de la siguiente ronda sea el que te sobró de la anterior más el que se sumó por bonificación de ronda completada. Este tiempo será siempre el mismo, cinco minutos. Además de una bonificación de tiempo extra también se obtendrá una bonificación de puntos.

Si el juego no proporcionara más ayuda que la bonificación extra de tiempo por ronda completada, este juego se haría bastante complicado, pero, aparte de esta bonificación, el juego está provisto de una serie de extras que facilitan la misión del jugador.

Al estar los movimientos de los androides vigilados por satélites, el juego cuenta con un mapa de la zona en la que se encuentra el jugador, y en el que se señala la posición del usuario y de los androides. De esta manera los androides están siempre localizados y no resulta tan difícil encontrarlos. Al iniciar una nueva partida, y al pasar de ronda, el mapa toma como centro la posición del usuario del juego.

Además de este mapa, ocasionalmente y de manera aleatoria aparecerán marcados en el mapa, una serie de ítems (objetos) que ayudarán temporalmente al jugador. La manera de obtener los beneficios que cada uno aporta es similar a la captura de un androide. Tienes que localizarlo con el terminal y pulsarlo en la pantalla del mismo estando a una distancia apropiada.

A continuación se describe el funcionamiento de estos:

- Parar el tiempo. Como ya hemos comentado con anterioridad, existe un ítem que permite detener la cuenta atrás, de tal forma que el jugador tenga unos segundos más para cumplir su misión. La duración de este efecto será de unos 45 segundos.

- Congelar androides. Este ítem permite paralizar temporalmente a todos los androides de una misma ronda, de esta manera el jugador goza de

una ventaja adicional para completar la ronda durante un tiempo de unos 45 segundos. Este ítem puede parecer muy similar al anterior pero en realidad da una ventaja mayor puesto que al parar el tiempo los androides siguen en movimiento y estos son bastante resbaladizos.

- Puntos x2. Si durante el tiempo que dura el efecto de este ítem el usuario realiza alguna acción que conlleve la suma de puntos al contador, este incrementará el valor de la acción sumándose el doble del valor original de ésta. El efecto temporal de este ítem asciende al minuto y medio.

Todos los objetos, una vez que aparecen en el mapa, tienen un tiempo límite de 30 segundos para encontrarlos y recogerlos. El tiempo que tienes para disfrutar de sus beneficios como hemos podido comprobar varía en función del ítem.

La intención era tener una diversidad de objetos mucho mayor que se alejara de la monotonía y dotara al videojuego de un carácter más lúdico, pero debido a que se trataba de contenido extra que se añadía con objeto de acercar la aplicación a lo que se entiende por videojuego, se decidió implementar las mínimas funcionalidades.

Una vez definido el videojuego que servirá para desarrollar el proyecto integrando el otro que se ha desarrollado paralelamente, cabe hacer una mención de las partes que afrontará con más énfasis el proyecto que esta memoria expone.

Concretamente las funcionalidades que abarca y desarrolla este proyecto son:

- El desarrollo de toda la lógica del juego. Esto abarca diferentes aspectos como la inteligencia artificial de los androides, el número de androides que hay por ronda y el paso de ronda, las bonificaciones de las que se beneficia el usuario, la aparición de los ítems y el control de sus efectos, además de otros contenidos extras.

- El manejo del GPS para obtener la localización del usuario al inicio del videojuego, con lo que todo esto conlleva. Además del manejo de la precisión del GPS y el control del error que este introduce. Evitando, en un principio, el inicio del juego hasta que no exista un precisión lo suficientemente aceptable y, durante el desarrollo de la partida, la perdida de precisión que en ocasiones se torna intolerable.

- El manejo del API de Google Maps para generar una vista de un mapa de la zona en el que el jugador se encuentra. Además, para enriquecer la partida con la librería de Google, se permite el manejo de diferentes funcionalidades como la de posicionar los androides, moverlos por el mapa y quitarlos, de igual manera que se puede hacer con los ítems.

- Por último, integrar la librería que se implementa en el proyecto de fin de carrera que se desarrolla paralelamente a éste, y que se usará para la introducción en este proyecto de los sensores de orientación, los acelerómetros y la inclusión de gráficos 3D, además de efectos de audio y control de eventos táctiles sobre la pantalla, todos ellos desarrollados en dicho proyecto.